Advanced Natural Language Processing Real-Time Speech Translator Application Design Document

Version: 1.0
Date: June 19, 2025
Developer: Nikita Petrucek 20248680
Project Type: AI/NLP Real-Time Translation System
Target Platform: Desktop Application

1. Title Page

1. Application name: Flowl
2. Application Title: Advanced Real-Time Speech Translator with Contextual Understanding
3. Version: 1.0
4. Date: 19/06/2025
5. Developer Nikita Petrucek
6. Project Type: AI/NLP Systems Programming Application
7. Target Platform: Desktop application

2. Table of Contents

3. Introduction/Overview

Purpose

The application specializes in providing subtitle-style translations that appear over existing content, making foreign language websites, videos, and applications instantly accessible. By combining efficient speech recognition with smart translation algorithms, it enables users to understand content in real-time without disrupting their workflow or requiring separate translation windows. The system's modular architecture ensures easy maintenance and updates while keeping the core functionality lightweight and responsive.

Scope

• Capture live audio from microphone (or other input sources)

• Segment audio intelligently using pauses, time-based triggers, and adaptive models

• Perform streaming speech-to-text (ASR) processing

• Parse recognized text into tokens with grammatical metadata

• Weight tokens and decide when to trigger translation

• Translate pieces incrementally with template logic, lightweight dictionaries, and optional fallback to external translation libraries

• Build output phrases on the fly, handle placeholders (e.g., missing subject), reorder words according to target-language syntax

• Validate or refine via back-translation or heuristics

• Display subtitle overlays or output synthesized audio (TTS)

• Provide comprehensive logging, configuration, and adaptation capabilities

Target Audience of Users

The Advanced Real-Time Overlay Translator serves desktop users who need quick, unobtrusive translation assistance during their daily computer activities:

Web Browser Users

Individuals browsing foreign language websites, reading international news, or accessing global e-commerce platforms benefit from real-time subtitle overlays that translate content without requiring page navigation or separate translation tools. The application seamlessly integrates with popular browsers to provide instant comprehension of web content.

Media Consumers and Streamers

Users watching foreign language videos on platforms like YouTube, Netflix, or streaming services can access real-time subtitles in their preferred language. The overlay functionality works across different media players and web-based video platforms, enhancing entertainment and educational content consumption.

Remote Workers and Students

Professionals participating in international video conferences, webinars, or online courses utilize the application for real-time translation during virtual meetings. Students accessing foreign language educational content or participating in international online programs benefit from immediate translation support without disrupting their learning environment.

Casual Desktop Users

Everyday computer users who occasionally encounter foreign language content during routine desktop activities appreciate the lightweight, always-available translation capability. The application runs quietly in the background, activating only when needed, making it ideal for users who want translation support without system overhead.

3.1. Application Overview

Flowl is a lightweight, desktop-focused NLP solution designed to provide seamless real-time translation for everyday computer interactions. This efficient application serves as a practical tool for breaking language barriers during web browsing, video streaming, online meetings, and other desktop activities where immediate translation is needed. Unlike resource-heavy translation platforms, this solution prioritizes minimal system impact while delivering accurate, contextual translations through an intelligent overlay system. It works by combining heuristic grammar-based logic with selective use of existing ASR, parsing, and translation libraries. It avoids reliance on large end-to-end neural translation models for the core real-time path, instead using an explainable, modular approach that provides transparency and adaptability.

4. Application Concept

Key Objectives

• Low latency: Translate and display subtitles almost immediately as the speaker talks

• Explainability: Each translation step is traceable - why a certain word or form was chosen, how the phrase structure was built

• Resource efficiency: Minimize heavy model use; rely on lightweight libraries or small local models for ASR and parsing; core translation logic is rule-based with optional small predictive components

• Adaptability: Support multiple language pairs by plugging in appropriate parsing/inflection libraries and template dictionaries. Enable tuning of pause thresholds or token weights

• Modularity: Clear separation of audio capture, ASR, parsing, weighting, translation logic, sentence building, validation, and UI/overlay components

4.1. Core Functionality and Innovation

The application's core innovation lies in its contextual translation triggering system, which analyses speech patterns, syntactic structures, and semantic content to determine when sufficient context exists for accurate translation. This approach eliminates traditional delays associated with waiting for complete sentence formation, enabling more natural conversational dynamics.

Key innovations include:

• Dynamic context enhancement that continuously refines translations as additional context becomes available
• Template-based translation with intelligent fall-back mechanisms
• Adaptive pause detection using mini-networks for threshold adjustment
• Explainable AI approach where each translation decision can be traced and understood

4.2 Use Cases / User Stories

• Live conversation assistance: A user speaks in language A; the other party sees subtitles in language B on screen or hears audio via headphones

• Lecture or presentation translation: Speaker's words are transcribed and translated in real time for an audience

• Video overlay: The application hooks into audio or video streams and shows translated subtitles on top of video playback

• Offline/edge use: In contexts without reliable internet, the app runs locally using lightweight models and rule-based translation logic

• Customization/domain-specific tweaks: Users can adjust token weights or provide domain-specific dictionary entries to improve translation accuracy in particular fields (e.g., medical, technical terms)

5. UX and UI

User Interface Goals

The interface design prioritizes minimal distraction while maintaining full functionality. Key design principles include:

• Minimal distraction: Overlay subtitles should be legible but not obstruct critical screen areas
• Extensive customization: Font size, color, position, opacity, background shading options
• Intuitive feedback & controls for start/stop capture, language selection, and settings adjustment
• Real-time diagnostic capabilities with confidence indicators and processing status
• Accessibility features including keyboard shortcuts, adjustable text size, and color-blind-friendly palettes

Primary Interface Components

• Main overlay/subtitle area: Translucent panel typically positioned at bottom of screen displaying translated text

• Control toolbar: Minimizable interface with essential controls (start/stop, settings, language selection)

• Settings dialog: Comprehensive configuration interface with tabs for Audio, Translation, Display, and Logging (if turned on)

• Diagnostic panel: Real-time view of processing pipeline status, token weights, confidence scores (if turned on)

• Notification system: Visual cues for translation events, latency alerts, and system status

6. Core Processing Modules

The backend architecture is organized into clear, modular components. Each module exposes a defined interface and communicates via structured data formats. This modular approach ensures maintainability, testability, and extensibility.

Audio Capture & Buffer Controller

Responsibilities: Read audio from microphone; maintain a rolling buffer; detect silence/pauses/time thresholds; trigger "segment ready" events.

- Inputs: Raw audio stream from selected input device
- Outputs: Audio snippets (PCM frames) handed off to ASR module
- Features: Device enumeration, noise suppression, gain control, silence detection
- Interface: Configurable sample rates, buffer management, and VAD integration

ASR Engine Wrapper

Responsibilities: Take audio snippets and produce recognized text with confidence scores; support incremental output when available.

- Primary Engine: OpenAI Whisper with streaming optimizations
- Alternative Options: Vosk for streaming, whisper.cpp in chunked mode
- Interface: asr_engine.process(audio_chunk) → {text: str, confidence: float, timestamp: ...}
- Features: Model selection based on performance requirements, adaptive quality settings

Parser / Token Analyzer

Responsibilities: Parse recognized text into tokens with comprehensive metadata including lemma, POS tags, dependency roles, and morphological features.

- Primary Library: spaCy for English and major languages
- Possible additional Tools: Stanza for specialized language support, pymorphy2 for Russian language
- Interface: parser.parse(text: str) → List[TokenInfo]
- TokenInfo Structure: {text, lemma, pos, dep, tense, gender, number, confidence}
- Features: Multi-language support, contextual analysis, syntactic structure extraction

Token Weighting & Trigger

Responsibilities: Assign numeric weights to tokens based on POS and metadata; decide when sufficient context exists for translation.

• Weighting Logic: Main verbs receive higher weights, pronouns moderate weights, particles low weights
• Threshold Management: Configurable weight thresholds with adaptive adjustment capabilities
• Interface: Accumulates weight_sum per segment; provides callbacks when translation should be triggered
• Features: Real-time weight visualization, adaptive threshold tuning, forced timeout mechanisms

Translation Core (Template-Based + Dictionary + Fallback)

Responsibilities: Generate candidate translation fragments using multiple approaches for optimal accuracy and speed.

Translation Mechanisms:
• Template Dictionary: Maps lemma+tense combinations to target-language fragments (e.g., Russian "продавал" → English ["have", "been", "selling"])
• Reordering Rules: Syntax-aware positioning based on source-target language differences
• Placeholder Logic: Intelligent insertion of subjects/objects when grammatical elements arrive out of order
• Predictive Submodule: Small model or heuristic for guessing missing elements based on context
• Fallback Integration: LibreTranslate, DeepL, or MarianMT for unknown terms

Interface: translate_fragment(token_infos: List[TokenInfo]) → TranslationFragment
Features: Multi-engine support, confidence scoring, contextual refinement

Sentence Builder / Incremental Assembler

Responsibilities: Integrate translation fragments into coherent target-language sentences with proper word order and grammatical structure.

• Assembly Logic: Manages partial sentence representation as ordered token list
• Reordering Engine: Handles late-arriving subjects, verb phrase positioning, modifier placement
• Placeholder Management: Maintains and resolves temporary placeholders as context develops
• Interface: Receives TranslationFragments sequentially; outputs assembled string when segment finalizes
• Features: Language-specific syntax rules, grammatical agreement checking, fluency optimization

Validation & Feedback Module

Responsibilities: Quality assurance through back-translation comparison and heuristic validation.

• Back-translation: Translate assembled sentence back to source language for similarity comparison
• Confidence Scoring: Generate reliability metrics for translation quality assessment
• Issue Detection: Flag potential mistranslations, missing context, or grammatical errors
• Interface: validate_translation(source_text: str, target_text: str) → ValidationResult
• Features: Similarity scoring algorithms, error pattern recognition, adaptive improvement suggestions

Display / Overlay Engine

Responsibilities: Render finalized translations as screen overlays or subtitles with professional presentation quality.

• Rendering Options: Transparent window overlays, video player integration, dedicated display modes
• Timing Management: Synchronized appearance/disappearance with speech timing
• Formatting Features: Text wrapping, positioning, font management, styling options
• Platform Integration: Desktop GUI frameworks (Tkinter, PyQt), video API hooks, OpenCV layers
• Interface: display_subtitle(text: str, style: SubtitleStyle, duration: Optional[float])

Configuration & Logging

Responsibilities: Comprehensive system configuration management and detailed operation logging.

• Configuration Management: User preferences, system parameters, language settings, performance tuning
• Logging System: Detailed records of recognition results, token analysis, weight calculations, translation fragments
• Diagnostic Tools: Real-time system monitoring, performance metrics, error tracking
• Interface: Configuration loaded at startup; dynamic UI adjustments; file/console logging
• Features: Export/import settings, log analysis tools, performance optimization recommendations

Adaptive Tuning Submodule (Optional)

Responsibilities: Continuous system improvement through usage data analysis and parameter optimization.

• Learning Mechanisms: Analyze logged data to identify improvement opportunities
• Parameter Adjustment: Dynamic tuning of token weights, pause thresholds, confidence requirements
• Model Training: Optional small ML components for context prediction and threshold optimization
• Interface: Periodic log analysis; automated parameter updates; user-approved improvements
• Features: A/B testing capabilities, rollback mechanisms, performance impact assessment

7. Visual Design

Design Language and Aesthetics: The application employs a clean, minimalist design approach with subtle visual elements that support functionality without creating distraction.

Interface Components and Layouts: Modular interface components enable flexible layout arrangements supporting different workflow preferences. Primary translation displays feature prominent text areas with clear distinction, while secondary controls remain accessible through intuitive navigation. Advanced features integrate seamlessly through progressive disclosure techniques that maintain interface simplicity for basic users.

Design Principles:
• Clean, minimalistic overlay that does not obstruct important content
• Semi-transparent backgrounds behind text to ensure readability regardless of underlying content
• Professional typography with legible sans-serif fonts and adjustable sizing
• Subtle visual hierarchy using color, size, and positioning
• Accessibility-first approach with high-contrast options and customizable elements

8. Audio Processing Design

Processing Goals & Requirements

The audio processing subsystem is a mission-critical component of the application, requiring precise control over real-time data flow, audio quality, and segmentation. Its primary function is to continuously capture user speech, identify logical speech boundaries, and ensure that downstream modules receive coherent, timely input.

Primary Objectives:

• Sub-500ms end-to-end processing delay from speech to on-screen translation
• High accuracy and robustness across variable microphone conditions and noisy environments
• Efficient preprocessing and encoding, tailored to the requirements of the ASR engine
• Adaptive performance tuning based on system resource usage
• Multi-threaded, non-blocking architecture to prevent audio dropouts and maintain responsiveness

Audio Input Interface

The input interface supports:

• Device enumeration and intelligent fallback (e.g., switching to internal mic if USB mic disconnects)
• User-selectable sample rates (16 kHz for whisper.cpp, 48 kHz for high-fidelity engines)
• Dynamic device switching, with seamless continuation of capture
• Live quality feedback to warn users of input clipping, excessive silence, or background noise
• Support for multiple input sources, including system audio loopback for streaming use cases

Preprocessing Pipeline

Includes a suite of real-time enhancement modules:

• Noise Suppression — Using lightweight spectral subtraction or real-time filters (e.g., RNNoise, WebRTC NS)
• Gain Control — Automatic gain normalization to maintain optimal volume levels for ASR
• Voice Activity Detection (VAD) - Fast, low-complexity detection via amplitude or WebRTC-based models
• Buffer Management - Ring buffer structure ensuring low-latency access and precise segment boundaries
• Quality Monitoring - Real-time SNR (Signal-to-Noise Ratio) estimation and jitter analysis

Segmentation Logic: Hybrid Boundary Detection

One of the key challenges in real-time speech translation is determining when to split the audio stream into coherent segments for processing by the ASR engine. Traditional methods often fall short:

Why single-method segmentation fails:

• Fixed-time segmentation (e.g., every 2 seconds)
• Simple, ensures steady flow
• Often splits words or phrases in unnatural places
• Can reduce ASR accuracy due to loss of acoustic context

• High likelihood of chopping mid-sentence or misaligning subtitles

Pause-based segmentation (VAD or silence detection)

• Aligns better with natural phrase boundaries
• Speakers may talk continuously without clear pauses
• Micro-pauses (e.g., 100–200ms) may be hard to distinguish from word gaps
• Silence detection fails in noisy environments or with low-mic sensitivity

Why the hybrid approach is better:

This application uses a hybrid segmentation strategy combining the strengths of both techniques:

• Primary segmentation is driven by VAD/silence detection with a configurable pause threshold (e.g., 500-700ms)
• Fallback segmentation is triggered when a segment exceeds a maximum duration (e.g., 5 seconds), ensuring ASR latency remains bounded even during long, uninterrupted speech
• An adaptive micro-model (optional) dynamically tunes pause sensitivity based on the speaker's rhythm, previous recognition errors, and real-time latency constraints
• Context preservation is achieved by retaining a short overlapping window (around 300ms) between segments to avoid losing transition words

Advantages of this hybrid logic:

• Smarter segmentation: Aligns with natural language units most of the time
• Fewer errors from mid-word cuts: Maintains ASR quality
• Predictable latency ceiling: Never wait indefinitely for a pause
• Modular adaptation: The segmentation module can be fine-tuned per user, language, or use case
• Future extension: Logged segmentation outcomes can train a lightweight ML module to improve thresholds automatically

Timestamp Preservation and Subtitle Sync

Each finalized audio segment is tagged with precise start/end timestamps for alignment with both recognized text and on-screen subtitles.
This enables real-time display sync, smoother subtitle transitions, and potential integration with media players or screen recording overlays.

2. Audio capture and preprocessing

Initialization & Setup

System Initialization Process:

• Audio stream configuration using libraries (PyAudio, sounddevice)
• Sample rate and frame size optimization (typically 20ms frames)
• Device capability detection and configuration validation

• Fallback device selection for improved reliability
• Performance parameter tuning based on system capabilities

Voice Activity Detection (VAD)

Advanced Voice Detection:
• Continuous frame analysis distinguishing speech from silence
• Multiple detection methods: energy threshold, WebRTC VAD integration
• Event generation: speech_start, speech_end signals for pipeline coordination
• Adaptive sensitivity based on ambient noise levels
• False positive reduction through temporal consistency checking

Adaptive Pause Thresholding

Intelligent Pause Management:
• Default threshold configuration (typically 500ms silence for phrase end)
• Adaptive mode: ML-based threshold adjustment using speech pattern analysis
• Feature inputs: average segment length, ASR confidence trends, speaking rate patterns
• Dynamic optimization: continuous improvement based on user interaction patterns
• Manual override capabilities for user-specific preferences

Buffer Management & Processing

Efficient Audio Buffer Handling:
• Ring buffer architecture for continuous audio capture
• State-based accumulation during speech detection
• Intelligent boundary detection combining silence duration and maximum segment length
• Per-segment preprocessing: noise reduction, normalization, format conversion
• Resource management: memory optimization and CPU usage monitoring

10. System Architecture & Data Flow

The application employs a sophisticated multi-threaded architecture ensuring smooth parallel operations across all major components.

Threading Architecture:
• Audio Capture Thread: Continuous microphone input and preprocessing
• ASR Processing Thread: Speech recognition with buffered input handling
• Translation Pipeline Thread: Token analysis, weighting, and translation logic
• Display Thread: UI rendering and overlay management
• Optional Features Thread: Extended functionality like learning modules and diagnostics

Data Flow Pipeline:
Audio Input → Buffer Management → ASR Processing → Token Analysis → Weight

Calculation → Translation Triggering → Fragment Translation → Sentence Assembly → Validation → Display Output

Inter-thread Communication:
• Queue-based message passing for reliable data transfer
• Event-driven coordination between processing stages
• Backpressure handling to prevent memory overflow during processing spikes
• Error propagation and recovery mechanisms throughout the pipeline

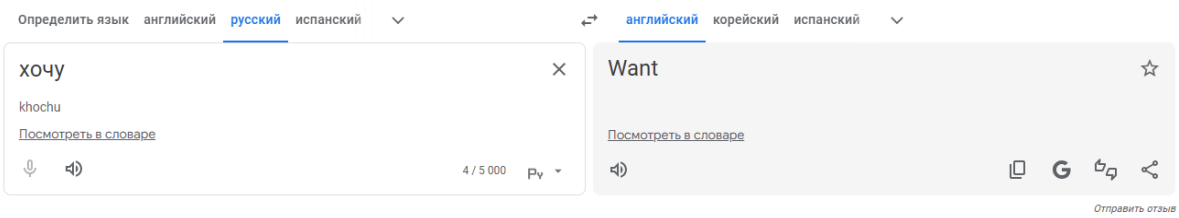
11. Translation example comparison

Let's take as an example Russian phrase "Хочу помидоры купить я" - translated to English will mean "I want to buy tomatoes".

Now we will try using two real-time translation technique – one is my pipeline, other – just translating the word by word without any manipulations.

First word:

Using no context approach we will get something like:



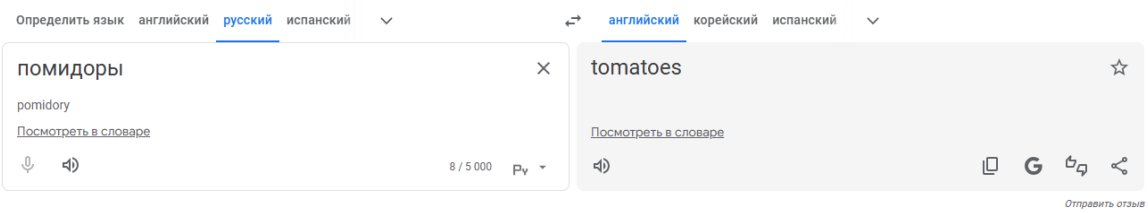
So, for word-by-word approach we simply write out:

Word by word sentence = [Want]

Now using our pipeline, we add some placeholders for clarity, and we get:

Pipeline sentence = ["Someone?" want "something?"]

Second word:

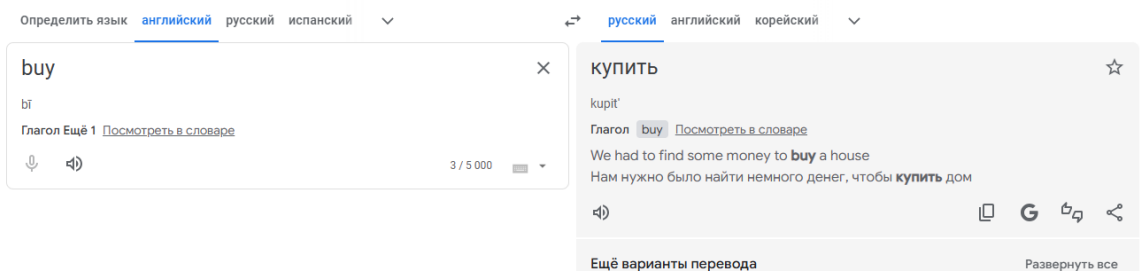


Word by word sentence = [Want tomatoes]

After pipeline we know that tomatoes is a noun, and we can replace the noun placeholder with our word:

Pipeline sentence = [“Someone?” want tomatoes]

Third word:

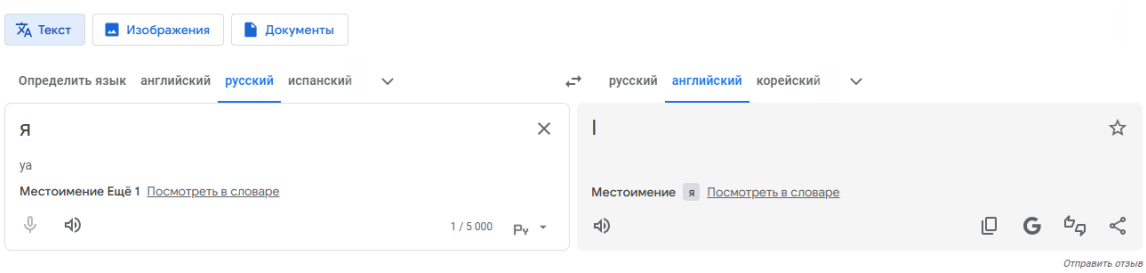


Word by word sentence = [Want tomatoes buy]

From pipeline we understand the context and therefore we can add particles regarding that context. Additionally, we place the verb before the noun as it should be.

Pipeline sentence = [“Someone?” want to buy tomatoes]

Last word:

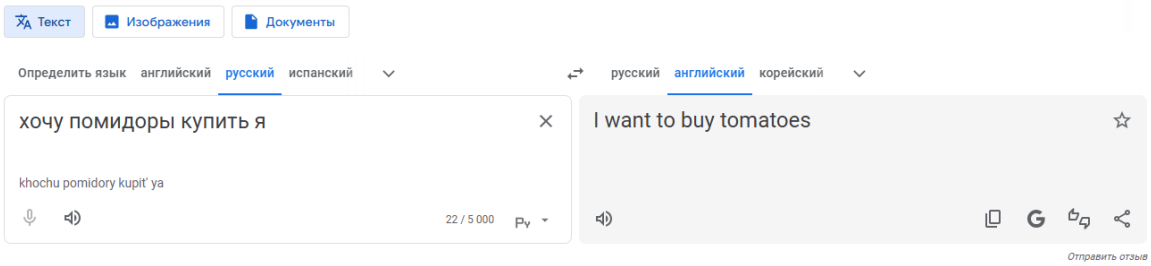


Word by word sentence = [Want tomatoes buy I]

For our last word we just replace it with a placeholder:

Pipeline sentence = [I want to buy tomatoes]

Validating using whole sentence translation:



The difference between word-by-word translation and application pipeline is even more noticeable in complex sentences, with past or future tenses.