# AI Keyboard

System-Level Intelligent Input Interface

Project Documentation — Final Round AI Hackathon
Team: Product Engineering | Track: AI Keyboard

## 1 Executive Summary

AI Keyboard transforms the passive keyboard interface into an intelligent, context-aware collaboration layer. Rather than forcing users to switch between their work and AI assistants, the system embeds AI directly at the point of input—providing real-time completions, voice input, and intelligent suggestions without interrupting workflow.

The system operates as a system-level input layer for Windows, understanding what application you're using, what you're trying to accomplish, and how to help. It integrates with Final Round AI to synchronize user profiles, resumes, job descriptions, and interview context, grounding all assistance in personalized, relevant suggestions.

Core Capabilities:

- Universal context awareness across all Windows applications
- Real-time inline completions with sub-200ms latency
- Keystroke pattern analysis with hesitation and correction detection
- Hybrid voice input with AI-enhanced transcription
- Cross-window context fusion for multi-document understanding
- MCP (Model Context Protocol) integration for extensible tools
- Speculative pre-generation for instant suggestions
- Adaptive personalization through style mimicry and LoRA adapters

## 2 Problem Statement

The keyboard is the most frequently used interface in modern knowledge work, yet it remains fundamentally passive. It captures keystrokes but understands nothing about intent, context, or desired outcomes. This creates significant friction:

- Users must context-switch between work and AI assistants, losing flow state
- Manual copy-pasting of context into chat interfaces wastes time
- AI suggestions lack awareness of what application is being used
- Voice input remains disconnected from keyboard workflows
- No system learns user preferences across applications
- Typing hesitation and corrections go unanalyzed—wasted cognitive effort

Research indicates context switching reduces productivity by up to 40%. We refer to this as the "Toggle Tax"—the hidden cost of switching between your work and AI tools. A 2-second thought becomes a 20-second chore.

**Our Solution:** Embed AI intelligence directly into the input layer, making assistance continuous, contextual, and invisible. The keyboard becomes a real-time collaborator that understands what you're doing and helps you do it better.

# 3 System Architecture

AI Keyboard employs a six-layer architecture designed for modularity, low latency, and extensibility. The system runs as a background process with the necessary privileges for global keyboard capture and window introspection.

## 3.1 Layer 1: Input Layer

The input layer captures all user input through four parallel subsystems:

**Keyboard Hooks:** Uses Windows low-level keyboard hooks (SetWindowsHookEx) to intercept all keystrokes system-wide with sub-millisecond latency. Maintains an input buffer with timing metadata to detect typing patterns—pauses indicate uncertainty, bursts indicate copying.

**Voice Engine:** Audio capture pipeline with Voice Activity Detection (VAD) and streaming transcription via Whisper API. Supports push-to-talk and continuous listening modes.

**Clipboard Monitor:** Tracks copy operations with semantic tagging, enabling intelligent paste operations that adapt content to target context.

**Gesture Capture:** Long-press spacebar activates AI suggestion deck; swipe gestures navigate alternatives and change tone modes.

## 3.2 Layer 2: Context Layer

The context layer implements a four-tier extraction pipeline:

| Tier | Latency | Extraction |
|---|---|---|
| Window Metadata | $<$5ms | Process name, window title, executable path, browser URL |
| UI Accessibility | $<$20ms | Focused element, surrounding elements, text selection |
| Visual OCR | $<$100ms | Screen capture + OCR for limited accessibility apps |
| Semantic Analysis | $<$50ms | Intent classification, entity extraction, tone inference |

**Cross-Window Context Fusion:** Beyond single-window analysis, the system detects relationships between multiple open windows using entity overlap analysis—shared companies, roles, skills, or dates indicate related windows.

## 3.3 Layer 3: Intelligence Layer

**Orchestrator:** Central coordinator managing request prioritization, context windowing, and response streaming. Implements debouncing to prevent API spam.

**Hybrid Inference:** Users choose between cloud inference (Claude API) for maximum capability, local inference (quantized Llama 3.1 8B via llama.cpp) for privacy, or hybrid mode.

**Local Inference Optimizations:** INT4 quantization reduces model size from 16GB to 5GB with 3x speedup. KV cache optimization cuts repeated inference by 60%. Speculative decoding uses a 1B draft model for 2-3x speedup.

**MCP Runtime:** Handles tool invocations through Model Context Protocol servers—calendar, email, database, and custom tools.

### 3.4 Layer 4: Learning Layer

**Writing Style Mimicry:** Pattern recognition model learns from user's historical messages, analyzing vocabulary, sentence structure, emoji patterns, and punctuation preferences.

**LoRA Adapter Personalization:** Lightweight fine-tuning that runs locally. Creates a 100MB adapter that modifies base model behavior to match user's writing style.

**Reinforcement Learning:** Implicit feedback signals improve suggestions over time based on accept/reject patterns.

### 3.5 Layer 5: Output Layer

**Ghost Text:** Inline suggestions displayed as semi-transparent text. Accept with Tab, dismiss with Esc.

**AI Deck Mode:** Long-press spacebar reveals card-based suggestions. Swipe to navigate alternatives and change tone.

**Command Palette:** Universal command interface (Ctrl+K) accepting structured and natural language commands.

### 3.6 Layer 6: Integration Layer

**Frai Sync:** Bidirectional synchronization with Final Round AI—imports resume, job targets, interview history.

**Session Memory:** Multi-tier memory preserving context across immediate, window, app, daily, and persistent levels.

## 4 Keystroke Intelligence Engine

A core innovation of AI Keyboard is the Keystroke Intelligence Engine—a real-time analysis system that transforms raw keyboard input into rich behavioral signals for AI assistance.

### 4.1 Session-Based Keystroke Capture

The system captures keystrokes in application-aware sessions with the following data structure:

| Field | Description |
|---|---|
| app_name | Active application (e.g., "Visual Studio Code", "Microsoft Edge") |
| window_title | Specific window context (e.g., "key_recorder.py - Zenith") |
| start_time / end_time | Session timestamps for duration calculation |
| raw keystrokes | Complete input including special keys (<BACKSPACE>, <SHIFT>, <ESC>) |
| final_text | Reconstructed text after all corrections applied |
| total_keys | Total keystroke count |
| backspaces | Number of deletion keystrokes |
| corrections | Detailed log of each correction with timestamp and position |

### 4.2 Correction Analysis

The system tracks every correction in granular detail:

```
{
  "timestamp": "2026-01-19T23:37:31.730110",
  "position": 1,
  "deleted": "i",
  "reason": "backspace"
}
```

This enables calculation of key metrics:

**Correction Rate:** Percentage of keystrokes that are backspaces. High correction rates (>30%) indicate user uncertainty—an optimal moment for AI suggestions.

**Hesitation Patterns:** When a user types "hi", deletes it, types "hello", deletes it, types "buddy", deletes it, then finally types "how are you"—this reveals decision uncertainty. The AI can learn that the user is searching for the right greeting and proactively suggest alternatives.

**Position-Based Analysis:** Corrections at position 0 (complete deletion) indicate message abandonment. Corrections at the end indicate typo fixes. Mid-text corrections suggest rephrasing.

### 4.3 Intent Detection from Keystroke Patterns

| Pattern | Detection Method | AI Response |
|---|---|---|
| High correction rate (>30%) | backspaces / total_keys | Offer suggestions proactively |
| Complete restarts | Position 0 deletions | Suggest alternative phrasings |
| Typing pause (>500ms) | Timestamp gaps | Pre-generate completions |
| Rapid burst typing | Low inter-key intervals | User is confident, stay passive |
| Repeated deletions of same word | Pattern matching on deleted text | Learn to avoid that suggestion |
| Punctuation hesitation | Multiple !!! then deleted | Suggest appropriate tone |

### 4.4 Cross-Application Context Switching

The session-based capture tracks application transitions:

```
Session 1: Microsoft Edge (16.67s) → typing message with corrections
Session 2: Microsoft Edge (3.24s) → tab navigation
Session 3: VS Code (0.17s) → ESC key (likely closing modal)
```

This enables:

- **Context Preservation:** When returning to an app, restore previous session state
- **Cross-App Intelligence:** If user was writing email about code, reference the code file they were editing
- **Workflow Detection:** Identify patterns like "research → write → code → test"

### 4.5 Real-Time Metrics Dashboard

Aggregated metadata provides session analytics:

| Metric | Value (Example) |
|---|---|
| Total sessions | 3 |
| Total keystrokes | 55 |
| Total duration | 20.08 seconds |
| Effective typing rate | 2.74 keys/second |
| Correction rate (Session 1) | 34% (18 backspaces / 53 keys) |
| Platform | Windows |

# 5 Key Features

## 5.1 Universal Context Awareness

The system understands any application without per-app configuration. Context awareness enables appropriate behavior adaptation: formal suggestions for email, technical precision for code, STAR structure for interview responses, casual register for messaging.

## 5.2 Speculative Pre-Generation

Rather than waiting for explicit user requests, the system begins inference when typing pauses are detected (default 300ms threshold). Three suggestions generate in parallel. When the user triggers suggestions, results are already ready—reducing perceived latency to near-instant.

## 5.3 Hesitation-Aware Suggestions

When the Keystroke Intelligence Engine detects high correction rates or repeated restarts, the AI proactively offers alternatives. Example: User types "hi" → deletes → "hello" → deletes → "buddy" → deletes. AI surfaces: "Having trouble with the greeting? Try: 'Hey there!', 'Good morning!', 'Hope you're doing well!'"

## 5.4 Adaptive Tone Control

Four instant-switch tone modes: formal, casual, polite, and funny. The system also learns from correction patterns—if a user frequently deletes exclamation marks, it reduces enthusiasm in suggestions.

## 5.5 Voice Input

Complete speech-to-text workflow: audio capture → VAD segmentation → Whisper transcription → AI cleanup (punctuation, filler removal). Target latency under 300ms.

## 5.6 MCP Tool Integration

Model Context Protocol enables extensible tool integration: calendar, email, database queries, and user-deployable custom servers.

### 5.7 Clipboard Intelligence

Maintains semantic clipboard history with content type detection, source window tracking, and similarity search via embeddings.

## 6 Final Round AI Integration

### 6.1 Plugin Architecture

The plugin declares permissions (profile read, resume read, interview history), capabilities (system keyboard, voice input, context extraction), and lifecycle hooks (interview start/end, job view).

### 6.2 Personalization Grounding

All suggestions incorporate Frai context. Interview responses retrieve relevant experiences from parsed resume. Job application assistance references target role requirements.

### 6.3 Interview Mode

When Frai detects an interview starting: suggestions pre-generate for common questions, STAR prompts appear automatically, stealth UI minimizes visibility during screen share, voice transcription captures interviewer questions.

## 7 Technical Implementation

### 7.1 Technology Stack

| **Component** | **Technology** |
|---|---|
| Runtime | Electron 28+ |
| UI Framework | React 18 + TypeScript |
| State Management | Zustand |
| Local Database | SQLite via better-sqlite3 |
| Vector Search | hnswlib-node |
| Native Modules | Custom Node addons for Win32 APIs |
| Cloud AI | Composer Based Architecture Models for Fast Inference |
| Local AI | llama.cpp with quantized Llama 3.1 8B |
| Voice | OpenAI Whisper API + Silero VAD |
| Keystroke Analysis | Python + JSON session logging |

| Metric | Target | Rationale |
|---|---|---|
| Memory usage | <200 MB | Background process constraint |
| CPU (idle) | <1% | No battery impact |
| Keystroke latency | <5 ms | Imperceptible delay |
| Suggestion latency | <200 ms | Maintains typing flow |
| Correction analysis | <10 ms | Real-time pattern detection |
| Voice transcription | <300 ms | Natural conversation pace |

| Data Type | Storage | Transmission | Retention |
|---|---|---|---|
| Keystrokes | Memory + local JSON | Cloud (opt-in) or local only | Session |
| Correction logs | Local encrypted | Never transmitted | Configurable |
| Voice audio | Memory only | Whisper API (TLS) | None |
| Session metadata | Local SQLite | Frai (opt-in) | Permanent |

### 7.2 Performance Targets

## 8 Security and Privacy

### 8.1 Data Handling

### 8.2 Privacy Mode Toggle

One-click privacy control (Ctrl+Shift+K) instantly disables all AI features and keystroke logging. Visual indicator shows status.

### 8.3 Sensitive Content Protection

Automatic detection and protection for password fields, banking URLs, SSN patterns, and user-defined blocklist. Keystroke capture automatically disables in sensitive contexts.

## 9 Hotkey Reference

| Hotkey | Action |
|---|---|
| Tab | Accept current suggestion |
| Shift+Tab | Accept word-by-word |
| Esc | Dismiss suggestion |
| Ctrl+Space | Force trigger suggestion |
| Alt+1/2/3 | Select alternative suggestion |
| Long-press Space | Enter AI Deck mode |
| Ctrl+K | Open command palette |
| Ctrl+Shift+Space | Push-to-talk voice |
| Ctrl+Shift+K | Toggle AI Keyboard on/off |

## 10 Future Roadmap

**Platform Expansion:** macOS support via CGEventTap APIs, Linux via X11/Wayland.

**Mobile:** Android custom keyboard using Accessibility API, iOS keyboard extension.

**Advanced Keystroke Analytics:** Predictive models trained on correction patterns, typing speed optimization suggestions, fatigue detection.

**Team Features:** Shared templates, organizational knowledge bases, collaborative context.

## 11 Conclusion

AI Keyboard represents a fundamental shift in human-computer interaction—transforming the keyboard from passive input device to intelligent collaborator. The Keystroke Intelligence Engine analyzes not just what users type, but how they type—detecting hesitation, corrections, and uncertainty to provide proactive assistance at exactly the right moment.

By embedding AI at the point of input with deep contextual understanding, we eliminate context-switching overhead and enable continuous, invisible assistance. The keyboard becomes a tool that understands your goals and helps you achieve them—from the first keystroke to the final round.

*Don't just type. Flow.*

---

AI Keyboard — Final Round AI Hackathon Submission