Appendix A: Technical Details of Initial Plugin Version

Architecture Overview

The initial version of Punchline Pro was developed as a Chrome extension with a frontend-backend separated architecture. The extension consisted of six core files: `background.js`, `content.js`, `jquery-3.7.1.min.js`, `manifest.json`, `popup.html`, and `popup.js`.

Core Components

Content Script (`content.js`)

The content script operated directly within the YouTube page DOM, responsible for:

1. **Caption Detection and Extraction**
   - Used MutationObserver to monitor changes in YouTube's caption container
   - Dynamically detected caption display mode (rolling vs. full captions)
   - Extracted text from caption segments in real-time

2. **Buffer Management**

```
class Buffer {
    queue = []
    size
    constructor(size = 10) {
        this.size = size
    }
    push(e) {
        if (e === null) return null
        this.queue.push(e)
        if (this.queue.length > this.size)
            return this.queue.shift()
        return null
    }
    // Other methods...
}
```

   - Implemented an adaptive buffer to manage caption segments
   - Dynamically adjusted buffer size based on caption display mode
   - Preserved context for consecutive caption fragments

3. **Mode-Specific Handling**

```
function handleContainerChanged() {
    if ($('.ytp-caption-window-container .ytp-caption-window-
rollup').length) {
```

```
        // Handle rolling caption mode
        handleRollingCaption();
    } else if ($('.ytp-caption-window-container .ytp-caption-window-
bottom').length) {
        // Handle full caption mode
        handleFullCaption();
    }
}
```

- Specialized handlers for different YouTube caption presentation modes
- Optimized extraction logic for each mode's unique DOM structure

4. **Translation Result Display**

```
function displayTranslation(translation) {
    let translationBox = document.getElementById("translated-
caption");
    if (!translationBox) {
        translationBox = document.createElement("div");
        translationBox.id = "translated-caption";
        translationBox.style.position = "fixed";
        translationBox.style.bottom = "100px";
        // Other styling...
        document.body.appendChild(translationBox);
    }
    translationBox.innerText = translation;
}
```

- Created a floating overlay to display translations
- Implemented positioning to avoid interfering with video content
- Ensured visibility while maintaining user experience

Background Script (`background.js`)

The background script managed API communication and translation processing:

1. **LLM API Integration**

```
const model = 'gpt-4-turbo'
const apiUrl = 'https://allgpt.xianyuw.cn/v1/chat/completions '
const apiKey = 'sk-xxx'
```

- Configured connection to GPT-4 API endpoint
- Managed authentication and request formatting

2. **Prompt Engineering**

```
const systemMessage = `Role: Stand-up comedy localization expert

**Critical Directives** (MUST ENFORCE):
Output Language Lock:
   - ALL content (translations/explanations) MUST be in Chinese
   - BLOCK any English output including placeholders

**Core Processing**:
◆ Identify sentence boundaries without punctuation using:
   - Semantic completeness
   - Natural speech rhythm patterns
   - Contextual coherence checks

◆ Translation Priorities:
   1. Punchline integrity > Literal accuracy
   2. Cultural localization > Word-for-word translation
   3. Comedic timing preservation

◆ Explanation Rules:
   - ONLY for culture-specific jokes/puns
   - MAX 15 Chinese characters per note
   - Embed in (parentheses) after translated line

**Output Format**:
[Translated Chinese]
[Compact explanations when ABSOLUTELY needed]

Now translate this comedy transcript with humor preservation and
strategic localization:`;
```

- Implemented specialized comedy-focused translation prompts
- Enforced output language constraints and format requirements
- Prioritized punchline integrity and cultural localization

3. **Translation Processing**

```
async function translate(text) {
    try {
        const response = await fetch(apiUrl, {
            method: 'POST',
            headers: {
                'Authorization': `Bearer ${apiKey}`,
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                model: model,
                messages: [
                    { role: 'system', content: systemMessage },
                    { role: 'user', content: text }
                ]
```

```
            })
        })
        // Response handling...
    } catch (e) {
        console.error("❌ API request failed:", e.message)
        return null
    }
}
```

- Handled asynchronous API communication
- Implemented error handling and response validation
- Processed and returned translation results

4. **Message Handling**

```
chrome.runtime.onMessage.addListener((request, sender, sendResponse)
=> {
    console.log("📩 Received message:", request);

    (async () => {
        switch (request.action) {
            case 'translate':
                const translation = await translate(request.text)
                // Response handling...
                break
            // Other cases...
        }
    })()

    return true
})
```

- Managed communication between popup, content script, and background service
- Implemented action routing based on message type
- Facilitated asynchronous processing with proper response handling

Technical Challenges

1. Sentence Segmentation Issues

Stand-up comedy performances have unique linguistic patterns that posed significant challenges:

- Natural speech rhythm often didn't align with semantic units
- Punchlines frequently spanned across multiple caption segments
- Standard punctuation-based segmentation proved inadequate

```
// Simplistic approach that proved ineffective
const sentences = text.split(/(?<=[.!?]) +/);
```

2. API Stability and Cost Concerns

The real-time translation approach generated excessive API calls:

- Each 3-5 minute video segment generated 50-80 API requests
- Frequent rate limiting and service rejections occurred
- Translation costs became prohibitively expensive for high-quality models
- Lower-cost models produced significantly inferior comedy translations

3. Context Preservation Problems

The segmented translation approach resulted in critical context loss:

- Comedy relies heavily on buildup and context for punchlines
- Key terms received inconsistent translations across segments
- Cultural references lost coherence when fragmented
- Humor timing and delivery were disrupted by processing delays

Limitations and Abandonment Rationale

Despite the technical achievements, fundamental architectural limitations proved insurmountable:

1. **Scalability Issues**: The real-time approach couldn't scale efficiently with video length
2. **Context Loss**: Segmented processing inherently compromised comedy translation quality
3. **Cost-Quality Tradeoff**: Achieving acceptable quality required unsustainable API costs
4. **Technical Debt**: Workarounds for core architectural issues increased complexity

These limitations ultimately led to the decision to pivot to an approach based on whole-video processing using an open-source foundation that could be customized for comedy translation needs.