

**School Of Computer Science and Engineering**

**Natural Language Processing and Computational Linguistics**

**CSI4001**

**Submitted To:**

**Dr. Sharmila Banu K**

**YouTube Context Analysis for Regional Language using API**

**Rishpraveen S: 21MID0151,** **Avantika Girigoswami: 21MID0011**

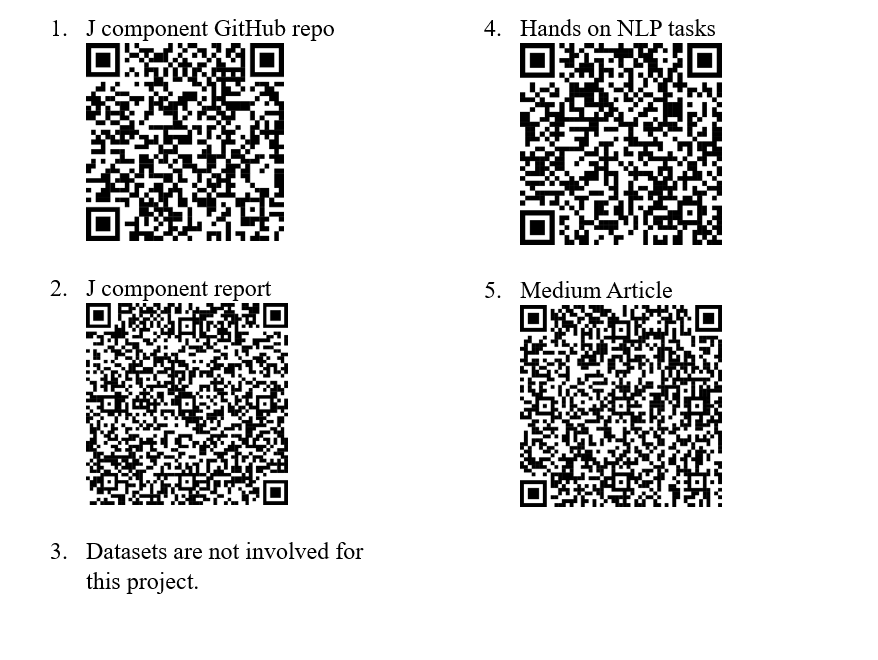
**M.Tech [Integrated]**

Class Number:VL2024250502123

Slot: C1+TC1

Department: Department of Database Systems

Winter Semester 2024-25



# Table of Contents

1. Abstract
2. Introduction
3. Problem Statement
4. Modules
5. Packages/API Used
6. Methodology with Architecture
   1. Architecture Overview
   2. Data Flow Description
7. Results and Discussion
8. Constraints/Limitations
9. Challenges Faced and How They Were Fixed
10. Code Implementation
    1. manifest.json
    2. background.js (Service Worker)
    3. popup.html & popup.js (User Interface)
    4. content.js (Content Script)
    5. options.html & options.js (Configuration)
    6. rules.json (Network Request Rules)
11. References

# Abstract

# With the exponential growth of online video watching, YouTube stands as a behemoth for information and entertainment. Nevertheless, within this content expanse come challenges in evaluating video content, analysing audience reception, and validating claims effectively. This project outlines building a Chrome Extension, the "YouTube Context Analyzer," that would make users have a greater contextual understanding of YouTube videos. Utilising the YouTube Data API and Google's Generative AI (Gemini), the extension retrieves video transcripts, analyses comment sections for sentiment, and provides on-demand fact-checking of highlighted text through a right-click context menu. By providing these capabilities in-browser, the extension hopes to better educate users, encourage critical viewing, and enable users to make more informed choices regarding the content they engage with.

# Introduction

# Web video sites, and especially YouTube, are at the heart of today's information creation and consumption. As valuable as they are in terms of educational and entertainment content, however, users struggle to get a sense of the fundamental message of a video, the general tone of the community debate, or the truthfulness of claims made. Basic metadata is provided by default by YouTube features, but more involved contextual analysis takes considerable manual labor. This project fills this void by developing a browser extension that embeds sophisticated analysis tools into the YouTube watching experience. The "YouTube Context Analyzer" offers features to: Try to fetch video transcripts for rapid content scanning, analyze sentiment distribution (positive, negative, neutral) in the comment section through AI, and Enable users to highlight text on the page or transcript and conduct an AI-driven fact-check. Using Google's YouTube Data API for data lookup and the Gemini API for advanced language understanding operations (sentiment analysis, fact-checking), the extension adds useful context not otherwise easily accessible, with the goal of a more informed and aware user experience.

# Problem Statement

YouTube viewers often have problems in effectively judging the content and context of videos:

* **Time-Wasting Content Evaluation:** Determining the most important information or points of a video takes viewing the entire video or fast-forwarding, which is inefficient. Having access to a transcript can make this greatly faster.
* **Knowing What's Being Said:** Reading manually through scores or thousands of comments to understand audience response is unfeasible. Users do not have a quick snapshot of the overall sentiment about a video.
* **Verifying Information:** Videos and comments can contain factual claims, misinformation, or opinions presented as facts. It takes verifying this information usually by navigating away from YouTube to search other sources, breaking the viewing experience.

There is a demand for an integrated tool that offers users easily accessible transcripts, summarizes sentiment of comments, and provides instant fact-checking functionality directly within the YouTube interface to solve these issues and encourage more critical consumption of online video content.

# Modules

The extension is built using a modular architecture:

1. **background.js (Service Worker):** The core processing unit.

* Loads and manages API keys (YouTube, Gemini) from storage.
* Listens for messages from the popup (popup.js) and context menu events.
* Coordinates API calls to YouTube Data API (fetch comment threads, list caption tracks) and Google Gemini API (sentiment analysis, fact-checking).
* Implements logic for fetching transcripts (attempting page scraping first, then API fallback).
* Processes data returned from APIs.
* Sends status updates and results back to the popup.
* Handles asynchronous operations using async/await.

1. **popup.html / popup.js (Popup UI):** The user interface displayed when the extension icon is clicked.

* Provides buttons to trigger transcript fetching and comment analysis.
* Displays status messages, processing indicators, and error notifications.
* Renders the fetched transcript text.
* Displays the results of the comment sentiment analysis (counts, sample comments).
* Displays the results of fact-checking requests initiated via the context menu.
* Listens for messages from the background script (background.js) containing results or status updates.

1. **options.html / options.js (Options Page):** (Code not provided but functionality implied)

* Provides a user interface (opened in a separate tab) for users to input and save their YouTube Data API and Gemini API keys.
* Uses chrome.storage.local (or sync) to securely store the API keys for use by background.js.

# content.js (Content Script):

* + Minimal script primarily used as a target for Chrome.scripting.executeScript from the background script.
  + Its main function in this implementation is to execute code within the context of the active YouTube tab to extract the current video ID from the page's URL (window.location.search).

# manifest.json (Configuration File):

* Defines the extension's properties, version, name, and description.
* Specifies required permissions (activeTab, scripting, storage, contextMenus, declarativeNetRequest).
* Declares host permissions required to interact with YouTube and Google API endpoints.
* Registers the background service worker (background.js).
* Defines the popup UI (popup.html).
* Registers the options page (options.html).
* Sets up the context menu item ("Fact-Check Selection").
* Declares the Declarative Net Request ruleset (rules.json).

# rules.json (Declarative Net Request Rules):

* + Contains rules to modify network requests. In this case, it includes a rule to potentially modify the Sec-Fetch-Site header for requests made from YouTube pages, which might assist in certain types of data fetching, although its specific impact requires detailed testing.

# Packages/API Used

* **Chrome Extension APIs:**
  + chrome.runtime: For messaging between scripts (sendMessage, onMessage) and managing the extension lifecycle (onInstalled).
  + chrome.storage: For storing API keys (local.get, local.set).
  + chrome.scripting: For executing scripts in the context of the active tab (executeScript).
  + chrome.tabs: For querying active tab information (query).
  + chrome.contextMenus: For creating and managing the right-click context menu item (create, onClicked).
  + chrome.declarativeNetRequest: For modifying network requests based on predefined rules.

# Web APIs:

* + fetch: For making HTTP requests to external APIs (YouTube, Gemini).
  + URLSearchParams: For parsing URL query parameters (to get video ID).
  + DOM APIs (document.querySelector, window.location): Used within executeScript to access page data.
  + JSON.parse, JSON.stringify: For handling JSON data from APIs.

# External APIs:

* + **YouTube Data API v3:**
    - commentThreads.list: To fetch top-level comments and replies for a video.
    - captions.list: To list available caption tracks for a video (used as a fallback for transcript fetching).

# Google Generative AI API (Gemini):

* + - models/gemini-1.5-flash-latest:generateContent: Used for comment sentiment analysis (balancing speed and cost).
    - models/gemini-1.5-pro-latest:generateContent: Used for fact-checking (leveraging stronger reasoning capabilities).

# Methodology with Architecture

* 1. **Architecture Overview**

The extension employs an event-driven architecture common in Chrome extensions, centred around a background service worker (background.js) that coordinates actions and communicates with the popup UI and external APIs.

**Initialisation:** On installation or startup, the background script loads API keys from chrome. storage and sets up the context menu.

# User Interaction (Popup):

* + - The user clicks the extension icon, opening popup.html.
    - popup.js sends a message (e.g., getTranscript, analyzeComments) to background.js containing the action and the active tab ID.

# User Interaction (Context Menu):

* + - The user selects text on the YouTube page.
    - The user right-clicks and selects "Fact-Check Selection".
    - The chrome.contextMenus.onClicked listener in background.js triggers, sending a factCheckSelection message internally with the selected text.

# Background Processing:

* background.js receives the message.
* It sends status updates to the popup (updateStatus message).
* If needed (for transcript/comments), it injects a script via Chrome. scripting. Execute the script in the active tab (content.js context) to retrieve the video ID.
* **Transcript:** Attempts to fetch the watch page HTML, parse ytInitialPlayerResponse for caption track URLS, and fetch the transcript text. If parsing fails, it uses the YouTube Captions API to list available languages (but doesn't fetch the text).
* **Comments:** Uses the YouTube CommentThreads API (handling pagination) to fetch comments. It then batches comments and sends them to the Gemini API (Flash model) for sentiment analysis, parsing the structured JSON response.
* **Fact-Check:** Sends the selected text and a structured prompt to the Gemini API (Pro model), requesting a JSON response with verdict, confidence, explanation, and sources.
* Handles potential errors during API calls or data processing.

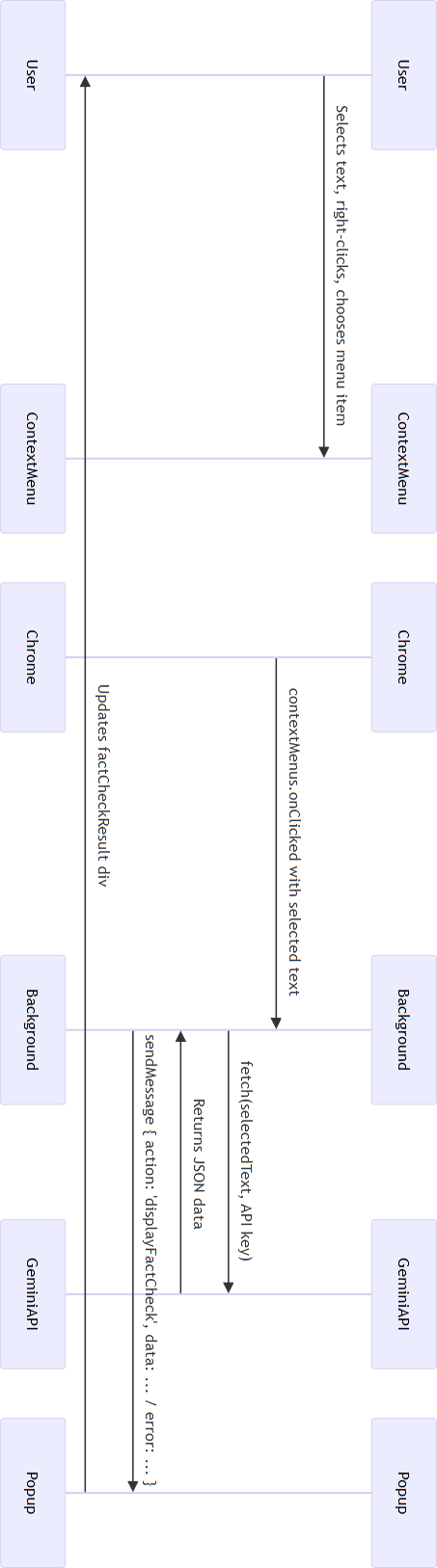
# Displaying Results:

* background.js sends the processed data or error messages back to popup.js using specific action messages (e.g., displayTranscript, displayCommentAnalysis, displayFactCheck).
* popup.js receives these messages and updates the corresponding sections in popup.html to display the results to the user.

# Data Flow Description

# 

*Fact-Check Flow:*

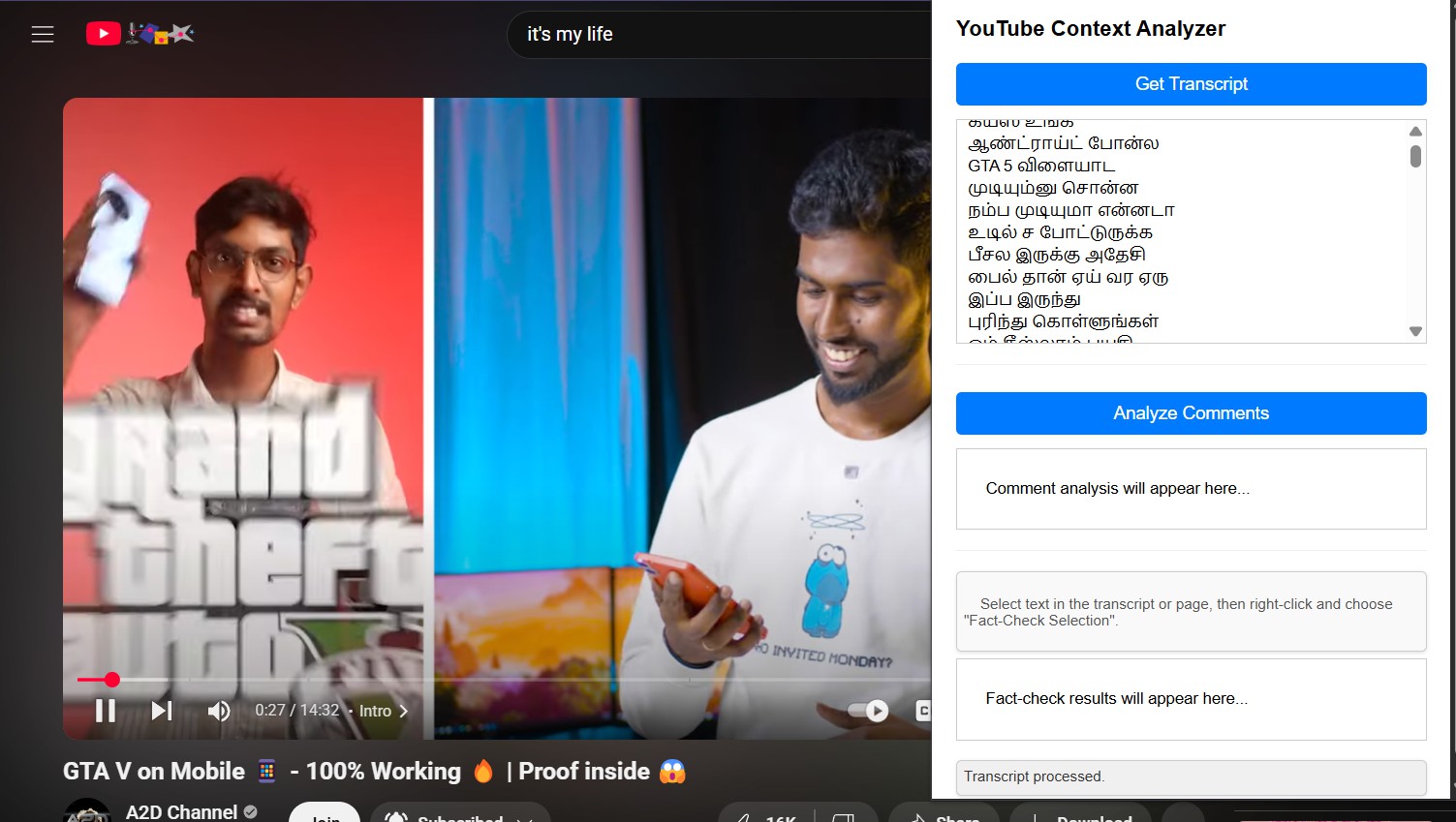
**

# Results and Discussion

*(Note: As no specific result screenshots were provided, this section describes the expected outcomes based on the implemented code.)*

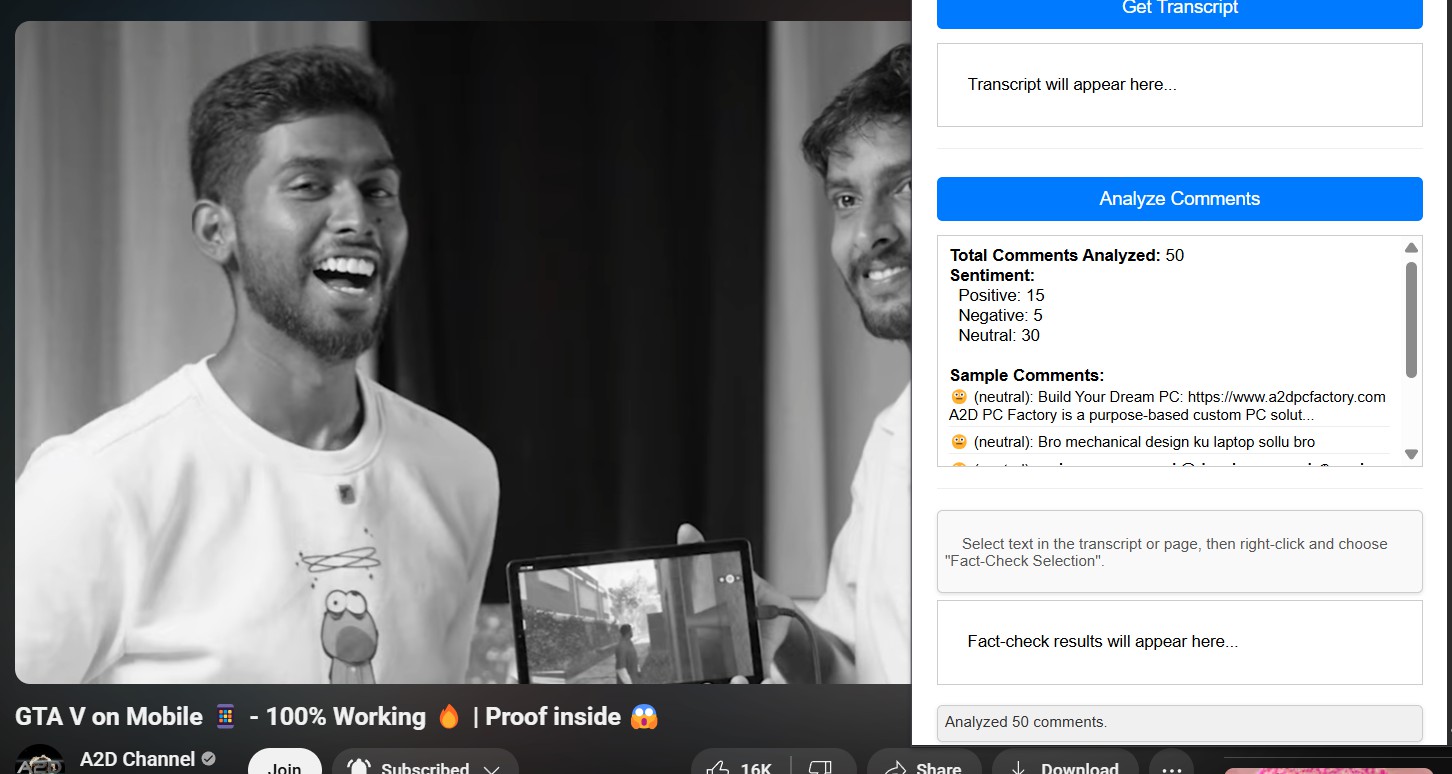
Upon successful execution, the extension provides the following results within its popup interface:

1. **Transcript Display:** The transcriptResult div is populated with the fetched transcript text. If fetched via page parsing, it includes the full text. If the fallback API method is used (due to parsing failure or lack of direct access), it displays a message listing the available caption languages found via the API, indicating that direct text download wasn't possible through that method. Errors during fetching are displayed clearly.



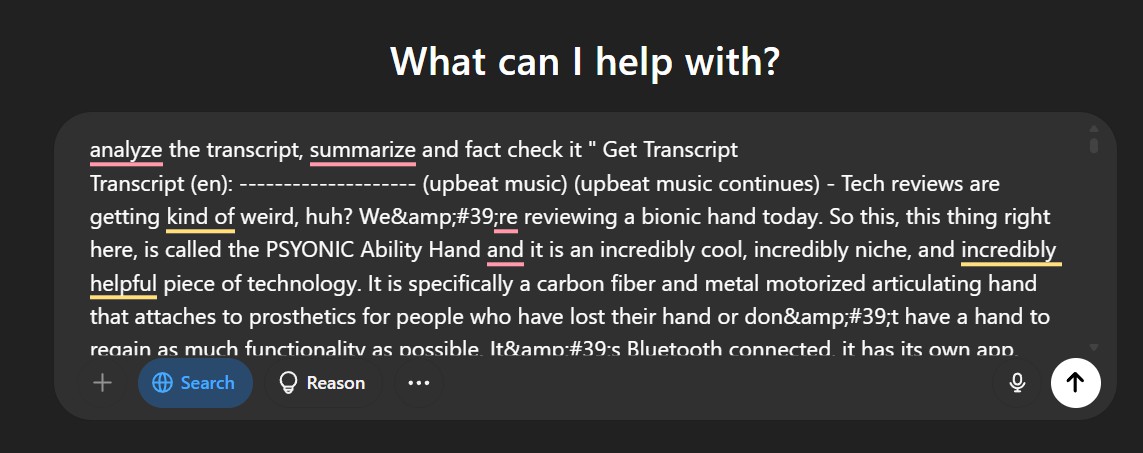
1. **Comment Analysis:** The commentResult div shows:

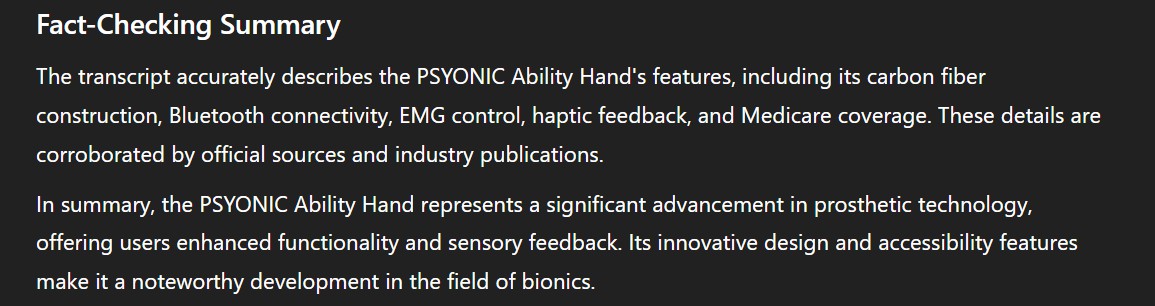
* The total number of comments fetched and analyzed.
* A breakdown of sentiment counts (Positive, Negative, Neutral).
* A sample (e.g., first 5) of the analyzed comments, prefixed with a sentiment emoji and showing the beginning of the comment text. This gives a qualitative feel alongside the quantitative summary. Errors during API calls or analysis are reported.



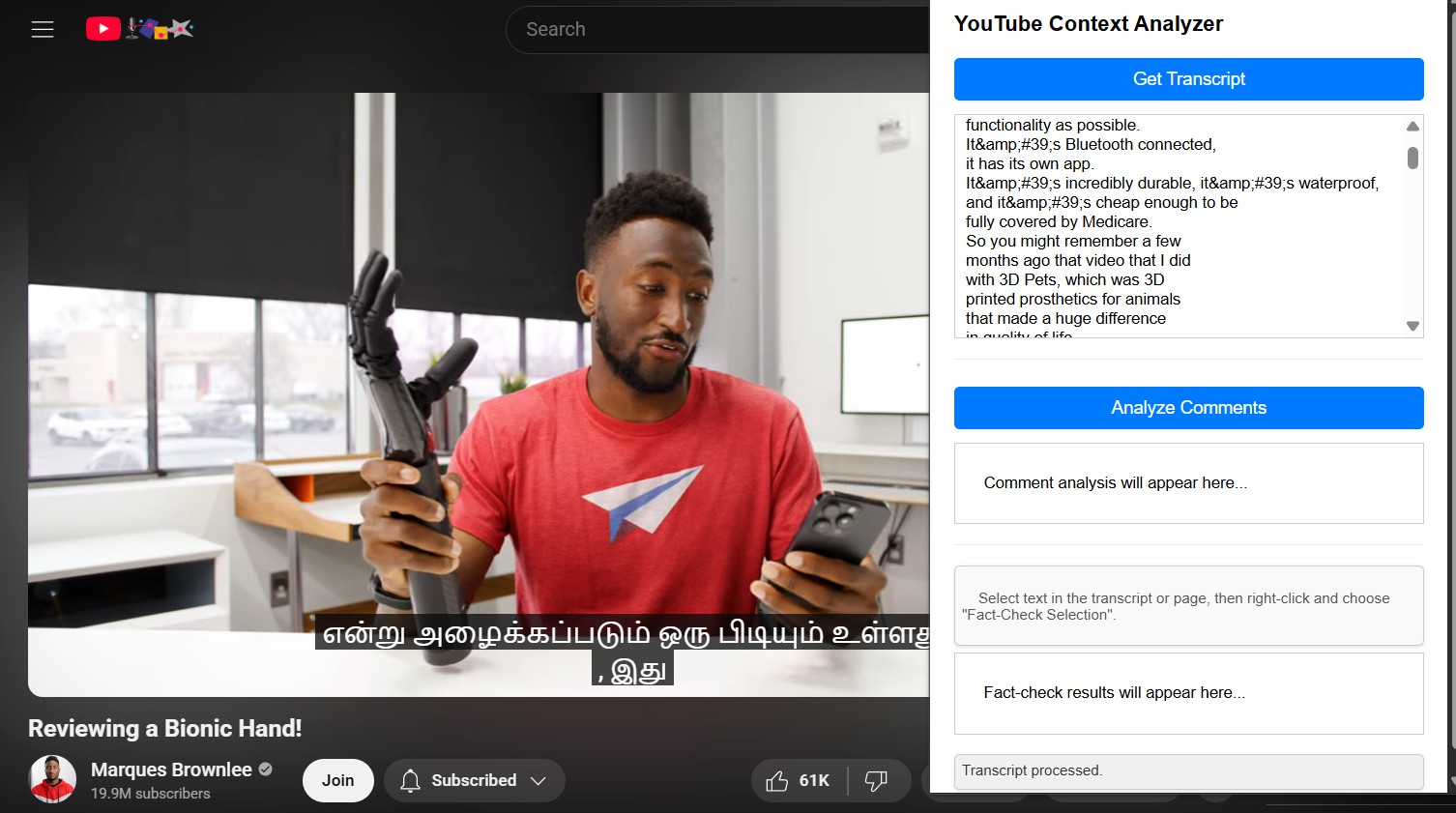
1. **Fact-Check Results:** When a fact-check is initiated via the context menu, the factCheckResult div displays the structured response from the Gemini API:

* **Verdict:** (e.g., "True", "False", "Misleading", "Unverifiable").
* **Confidence:** A percentage score indicating the AI's confidence in the verdict.
* **Explanation:** A concise rationale for the verdict.
* **Sources:** A list of potential source URLs or references provided by the AI (displayed as clickable links if possible). Errors are shown if the fact-check fails.

**This is not functional yet due to API limitations, but there is an alternate method to use the transcript as a Chatgpt or other generative AI models’ prompt and verify the facts**



A summary of the MKBHD’s youtube video



1. **Status Updates:** The status div provides real-time feedback on the extension's operations (e.g., "Fetching comments...", "Analyzing sentiment...", "Fact-check complete.", "Error: API Key invalid.").

# Discussion:

The add-on effectively brings several contextual analysis features into the YouTube environment. The transcript function facilitates rapid skimming of content. The comment analysis offers an invaluable, otherwise unobtainable, summary of community response. The fact-checking functionality provides an easy means of checking claims without exiting the page. The use of external APIs (YouTube, Gemini) makes the analysis strong but also adds dependencies and possible expense. The transcript retrieval strategy (page parse first) is a try to circumvent more rigid API requirements, but is less solid as a result than actual API approaches. That multiple Gemini models are used (Flash for comments, Pro for fact-checking) is a design choice that prioritizes

performance/cost over reasoning power. Overall, the tool supplies considerable value addition to those looking to delve deeper into context on YouTube.

# Constraints/Limitations

* **API Key Dependency:** Requires users to obtain and configure their own YouTube Data API and Gemini API keys via the options page. Functionality is blocked without valid keys.
* **API Costs & Quotas:** Use of YouTube Data API and Gemini API is subject to quotas and potential costs, which the user bears. Heavy usage might exceed free tiers.
* **Transcript Fetching Reliability:** The primary method of fetching transcripts by parsing page data (ytInitialPlayerResponse) is fragile and may break if YouTube changes its website structure. The API fallback only lists languages, not the text itself without potentially more complex authentication.
* **AI Accuracy and Bias:** The accuracy of sentiment analysis and fact-checking depends entirely on the Gemini models. These models can make errors, exhibit biases present in their training data, or struggle with nuanced/context-dependent language. Verdicts should be seen as informative aids, not definitive truths.
* **Limited Fact-Check Scope:** Fact-checking only operates on user-selected text snippets, not the entire video's audio/visual content. Context outside the selected text is missed.
* **Comment Analysis Scope:** Analysis is typically performed on a subset of comments (limited by maxResults and API pagination) for performance and cost reasons, not necessarily *all* comments on a very popular video.
* **Network Dependency:** Requires an active internet connection to communicate with Google APIs.
* **Browser Compatibility:** Designed as a Chrome Extension, likely compatible with other Chromium-based browsers (Edge, Brave) but not Firefox or Safari without modification.
* **Real-time Updates:** Analysis is triggered on demand (button click/context menu), not continuously monitoring in the background unless re-triggered.

# Challenges Faced and How They Were Fixed

1. **Dynamic Content Loading on YouTube:** YouTube heavily uses AJAX/SPA principles, meaning video changes or data updates don't always trigger a full page reload. Simple content scripts running only on page load would miss updates.
   * **Solution:** While not explicitly shown in the final code snippets for *this*

version, a common solution involves using MutationObserver in the

content script to detect relevant DOM changes (like the video player updating) and re-trigger data extraction. For this version, relying on user clicks in the popup for explicit re-analysis circumvents this for transcript/comments. The video ID is fetched dynamically on each request using scripting.executeScript.

1. **Extracting Video ID Reliably:** Getting the current video ID is crucial. Relying solely on the content script's initial load might fail if the user navigates within YouTube.
   * **Solution:** Implemented fetching the video ID *on demand* from the background script using Chrome.scripting.executeScript targeting the active tab. This ensures the ID corresponds to the currently viewed video when an action is triggered.
2. **Fetching Transcripts without OAuth:** Direct programmatic access to YouTube transcript text often requires OAuth 2.0 authentication, which is complex for a simple extension.
   * **Solution:** Employed a two-pronged approach:
     + Attempt to parse the ytInitialPlayerResponse object embedded in the YouTube watch page's HTML, which sometimes contains caption track URLs accessible without OAuth. This is less reliable but avoids complex authentication.
     + If parsing fails, fall back to using the YouTube Data API (captions.list) which only requires an API key but just lists available tracks, not the content. The result informs the user which languages are available.
3. **Structuring AI Prompts for Reliable Output:** Getting consistent and correctly formatted output (especially JSON) from Generative AI models like Gemini requires careful prompt engineering.
   * **Solution:** Developed specific prompts clearly outlining the task (sentiment analysis, fact-checking) and explicitly requesting the output *only* in a valid JSON format with predefined fields. Included instructions like "Respond ONLY with..." and used the responseMimeType: "application/json" parameter (where supported/effective) to further encourage correct formatting. Added try...catch blocks around JSON.parse to handle cases where the AI might still return malformed output.
4. **Handling API Errors and Rate Limits:** Calls to external APIs can fail due to invalid keys, network issues, quota limits, or server errors.

* **Solution:** Implemented robust error handling using try...catch blocks around fetch calls. Checked the response.ok status and attempted to parse error messages from the API response body. Displayed informative error messages to the user via the popup's status area instead of crashing.

1. **Managing Asynchronous Operations:** Multiple steps involve asynchronous operations (fetching keys, getting video ID, API calls). Coordinating these correctly is essential.

* **Solution:** Utilized async/await syntax extensively throughout the background.js script to manage asynchronous flows logically and avoid callback hell, making the code easier to read and maintain.

# Code Implementation

*(Includes the provided code snippets with explanations relevant to the final functionality)*

# manifest.json

{

"manifest\_version": 3,

"name": "YouTube Context Analyzer", "version": "0.2.0",

"description": "Extracts transcripts, analyzes comments, and fact-checks YouTube videos.",

"permissions": [ "activeTab", "scripting", "storage", "contextMenus",

"declarativeNetRequest"

],

"declarative\_net\_request": { "rule\_resources": [

{

"id": "ruleset\_1", "enabled": true, "path": "rules.json"

}

]

},

"host\_permissions": [ "\*://\*.youtube.com/\*", "https://\*.googleapis.com/\*"

],

"background": {

"service\_worker": "background.js"

},

"action": {

"default\_popup": "popup.html", "default\_icon": {

"16": "icons/icon16.png",

"32": "icons/icon32.png",

"48": "icons/icon48.png"

}

},

"options\_ui": {

"page": "options.html", "open\_in\_tab": true

},

"icons": {

"16": "icons/icon16.png",

"32": "icons/icon32.png",

"48": "icons/icon48.png",

"128": "icons/icon128.png"

}

}

* **manifest\_version: 3**: Specifies the current manifest version.
* **permissions**: Declares necessary permissions: activeTab (access current tab), scripting (inject scripts), storage (save API keys), contextMenus (add right-click menu), declarativeNetRequest (modify network requests).
* **declarative\_net\_request**: Links to rules.json for network request modification rules.
* **host\_permissions**: Grants access to YouTube pages and Google API endpoints.
* **background**: Registers background.js as the service worker.
* **action**: Defines the popup (popup.html) and icons.
* **options\_ui**: Specifies the options page (options.html) for API key configuration.

# background.js (Service Worker)

* keys loaded from storage.
* **loadAPIKeys**: Asynchronously retrieves API keys from chrome.storage.local.
* **updatePopupStatus, sendDataToPopup**: Utility functions for communicating with the popup UI.
* **getVideoIdFromTab**: Uses chrome.scripting.executeScript to run code on the page and extract the 'v' parameter (video ID) from the URL.
* **fetchTranscriptFromPage**: Attempts to fetch the transcript by parsing the ytInitialPlayerResponse from the video page's HTML. Includes logic to find appropriate language tracks. Crucially, it contains a fallback.
* **fetchAvailableTranscriptLangsAPI**: *Fallback function*. Uses the YouTube Data API (captions.list) to list available languages if page parsing fails. Note: This *lists* languages, doesn't fetch the actual transcript text via API in this implementation.
* **fetchAndAnalyzeComments**: Fetches comments using YouTube Data API (handles pagination). Batches comments and sends them to the Gemini API (Flash model) for sentiment analysis using a structured prompt. Parses the JSON response.
* **performFactCheck**: Called by the context menu listener. Sends selected text to the Gemini API (Pro model) with a prompt requesting a structured JSON output (verdict, confidence, explanation, sources). Parses the JSON response.

# Event Listeners:

* chrome.runtime.onMessage: Handles messages from the popup (getTranscript, analyzeComments, factCheckSelection) and routes them to the appropriate async function.
* chrome.runtime.onInstalled: Creates the context menu item on installation and performs an initial check for API keys.
* chrome.contextMenus.onClicked: Listens for clicks on the context menu item and triggers the fact-checking process by sending a message to its own message listener.

# popup.html & popup.js (User Interface) popup.html

* Basic HTML structure with buttons for actions (getTranscriptBtn, analyzeCommentsBtn).
* Divs to display results (transcriptResult, commentResult, factCheckResult).
* A status area (status) and processing indicator.
* Links to popup.css and popup.js.

# popup.js

* Gets references to HTML elements.
* **setProcessing**: Manages the visual state (disables buttons, shows indicator) during background operations.
* **updateStatus**: Updates the status message div.
* **displayFormatted... functions**: Format the data received from the background script into user-friendly HTML for display in the result divs.
* **Button Event Listeners**: Add listeners to the buttons. When clicked, they check if processing is ongoing, update the status, ensure the active tab is a YouTube video page, and then send the corresponding action message (getTranscript or analyzeComments) to the background script.
* **chrome.runtime.onMessage Listener**: Listens for messages from the background script (updateStatus, displayTranscript, displayCommentAnalysis, displayFactCheck). When a message is received, it calls the appropriate function to update the UI (status, results divs) and manages the processing state.

# content.js (Content Script)

console.log("YouTube Context Analyzer: Content script loaded (or injected).");

// This script primarily exists to be \*injected\* by the background script

// using chrome.scripting.executeScript() to get page-specific data like the video ID.

// It doesn't need much code itself initially, as the functions to execute

// are passed directly in the executeScript call from background.js.

// We \*could\* add listeners here if the background script needed to push

// UI changes \*to\* the page (e.g., highlight text), but let's keep it simple for now.

// Example of how it might listen for commands from background later: chrome.runtime.onMessage.addListener((request, sender, sendResponse) => {

if (request.action === "highlightText") {

console.log("TODO: Highlight text on page:", request.textToHighlight);

// Add DOM manipulation logic here sendResponse({ success: true });

}

});

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487). JavaScript IGNORE\_WHEN\_COPYING\_END

* This script is intentionally minimal in this version.
* Its primary role is to provide an execution context *within the YouTube page* for scripts injected by background.js (specifically, the function within getVideoIdFromTab that reads window.location.search).
* The example message listener shows how it *could* be used for more complex interactions (like highlighting text on the page based on commands from the background), but this is not implemented in the current core functionality.

# options.html & options.js (Configuration)

* **options.html**: Would contain HTML form elements (input fields) for the user to enter their YouTube Data API Key and Gemini API Key. It would also include a "Save" button.
* **options.js**: Would contain JavaScript to:
  + Load any previously saved API keys from chrome.storage.local when the options page is opened and populate the input fields.
  + Add an event listener to the "Save" button. When clicked, it would read the values from the input fields and save them to chrome.storage.local using chrome.storage.local.set({ youtubeApiKey: '...', geminiApiKey: '...' }).
  + Provide visual feedback to the user (e.g., "Keys saved!").

# rules.json (Network Request Rules)

[

{

"id" : 1,

"priority": 1,

"action" : {

"type" : "modifyHeaders", "requestHeaders": [

{ "header": "Sec-Fetch-Site", "operation": "set", "value": "same-origin" }

]

},

"condition" : {

"urlFilter" : "\*://\*.youtube.com/\*", "resourceTypes" : ["xmlhttprequest"]

}

}

]

IGNORE\_WHEN\_COPYING\_START

content\_copy download Use code [with caution](https://support.google.com/legal/answer/13505487). Json

IGNORE\_WHEN\_COPYING\_END

* This file defines rules for the declarativeNetRequest API.
* **id, priority**: Rule identification and execution order.
* **action**: Specifies modifying request headers (modifyHeaders). It targets the Sec-Fetch-Site header and sets its value to same-origin. This *might* influence how YouTube servers or intermediate proxies handle certain background requests (xmlhttprequest), potentially making some data fetching easier or more reliable in specific scenarios, but its exact effect can be complex and depend on YouTube's infrastructure.
* **condition**: Applies this rule only to xmlhttprequest resource types originating from YouTube domains (\*://\*.youtube.com/\*).

# References

* Chrome Developer Docs: <https://developer.chrome.com/docs/extensions/>
* YouTube Data API V3 Overview: <https://developers.google.com/youtube/v3/getting-started>
* Google AI for Developers (Gemini API): <https://ai.google.dev/>
* MDN Web Docs (Fetch API, DOM, etc.): <https://developer.mozilla.org/>