Kodi Architecture Enhancement

CISC 322

Assignment 3 Report

Tuesday, December 5, 2023

Group 8 : KidKodi
Aidan Gardner - (20agg2@queensu.ca)
Gavin Chin - (21gc9@queensu.ca)
Daniel Garami - (21deg1@queensu.ca)
Barkev Keyork Sarkis - (19bks5@queensu.ca)
Mahir Khandokar - (21mhk5@queensu.ca)
Adrian Putz-Preyra - (16app2@queensu.ca)

Table of Contents:

Abstract

This report presents a detailed proposal of an enhancement to Kodi's media player by implementing real-time speech-to-text capabilities. Our current system analysis of Kodi outlined its modular and layered architecture, which facilitates the proposed enhancement without major code refactoring. Two approaches are considered: a localized implementation using customizable deep learning models, and a cloud-based implementation using already implemented ASR systems. The chosen approach was determined by considering many different factors in relation to the enhancement, with us ultimately deciding on using a localized implementation for its advantages in customizability and future development potential. Use cases and plans for testing are outlined, along with potential risks related to cost, scalability, and data privacy. Lessons learned highlight the importance of maintenance, testing, performance optimizations, and stakeholder considerations in the implementation of AI-based enhancements.

1. Introduction

Over the past few months, our group has gained a comprehensive and detailed understanding of Kodi's functionalities and architecture. Through previous reports, we have studied the components of Kodi to develop a conceptual and concrete architecture as we delved into the details of the system. The purpose of this report is to propose a new feature that will add enhancement to the already existing functionality of the multimedia player.

With artificial intelligence continuing to take the world by storm, our group decided it would be a great time to jump in on the trend. As a result, our proposed enhancement aims to leverage artificial intelligence to provide for a better user experience, as we suggest the use of AI to generate real-time subtitles or lyrics based on speech input. In this report, we demonstrate two implementations of our proposed enhancement, each containing an explanation of any new interactions with existing components. We will use a SEI SAAM analysis to look at Kodi's various stakeholders and will use logical reasoning to weigh the advantages and disadvantages of the implementation strategies to decide which is best in relation to several NFRs. The report will also explore the influence of the newly introduced feature on both the high and low-level conceptual architectures of Kodi, while also looking at the directories and source codes affected by the feature's implementation. Subsequently, the report will delve into our strategy for testing the proposed feature and addressing the potential associated risks.

2. Current System

As explored in our previous analyses of Kodi's conceptual and concrete architecture, we have identified the modular and layered structure that is currently implemented within the system. This architectural style serves as the backbone of Kodi's versatility and can be split into four separate layers. These layers are all in close interaction with each other, and feature the Client Layer, Presentation Layer, Business Layer, Data Layer. Additionally, Kodi utilizes an assortment of common libraries across all layers. Due to Kodi's open-source nature, the developers utilized this structure to allow for teams to work independently on different areas without major reliance on other layers. As a result, Kodi's current architecture opens up the door to change without much of a need for re-factoring across all layers. This is only enhanced by Kodi's exceptional add-on integration for increased functionality. Our proposal aims to leverage these design techniques for an easy and efficient integration of a new feature.

3. Proposed Enhancement

3.1 Overview

Although Kodi already has the ability to leverage its add-on capabilities to download subtitles from various web sources, this only proves beneficial when the desired subtitles are available or in the wanted language. The reliance on a third-party source for subtitles is not ideal and limits the scope of programs that can use this add-on. As a result, our group has proposed utilizing artificial intelligence capabilities to allow for real-time speech-to-text generation in the context of subtitle or lyric generation. Not only does this mitigate the dependence on internet sources, but it also broadens the availability of subtitles and lyrics to include more languages, programs, or even use on live TV. Our proposed enhancement will add increased multilingual support, accessibility, inclusivity, and all the benefits stemming from real-time interaction.

3.2 Effects

The addition of real-time STT translation will provide a massive effect on system functionality with a focus on user experience. Users from across the world can now enjoy and better understand media content in their preferred language, breaking down language barriers and expanding the reach of content. In terms of effects on the already existing system, Kodi's architectural layout allows for an enhancement like this to be enabled with limited struggle. With all of Kodi's layers already in constant interaction with one another, there would be no need to re-factor the high-level conceptual architecture to incorporate our proposed enhancement as the data flow between layers is already suitable. In terms of the low-level conceptual architecture, the Data Layer provides us with needed network calling capabilities, and the other layers have already built frameworks for subtitle use. We already have the means to implement our enhancement whether it be through the use of plug-ins and addons or by using specific interfaces, and can leverage the already existing capabilities of the Player Core Module for seamless integration and smooth playback.

Now we will take a look at how the enhancement of real-time translation could potentially affect the maintainability, evolvability, and testability of the system. Starting with the effects of system maintainability, depending on how the enhancement is to be implemented will change maintenance requirements. Using a third-party API that has been trained for the purpose of real-time subtitle generation will minimize maintenance, however, increases the dependence on updates and resolution of issues regarding the API. On the other hand, the creation of a custom ML model may increase customization and flexibility but will require much more maintenance time if the model is being trained or updated with new data. Overall system evolvability will not be impacted too much as a result of this enhancement due to the implemented modular structure and specific nature of the enhancement, however, the evolvability specifically with the ML model has great potential. When adding or modifying the enhancement, there should not be any cases where the current features of Kodi are negatively impacted. As a mainly AI-based enhancement, testability will be difficult. This enhancement should work on a variety of languages, and programs, and as a result, needs to be tested on a large source of data. It should be able to translate and generate subtitles for different languages, tones, and vocal deliveries with a high degree. Within Kodi, this increases the time needed for testing as many factors must be considered when using this feature. Since Kodi's core functionality stems from its Player Core, the addition of subtitle generation must be scrutinized to ensure it does not negatively impact any existing features. Lastly, taking a look at system performance, real-time subtitle translation may introduce additional processing and overhead. To account for this, optimizations may be needed to ensure smooth playback without impacting system performance.

3.3 Interactions

Upon implementation of real-time subtitle generation, new interactions within the system arise. To begin with, whether it be through the use of a localized implementation or a third-party API, the actual translation capabilities will need to be in constant communication with mainly the Player Core Module. The chosen interface will provide us with our translation information and will be where all of the data is being generated and received. With this information in hand, we need to be able to display the subtitles in real-time. To do so, we would need to hook into the Player Core Module to ensure we can synchronize video and audio with the subtitles. Additionally, a new interaction would be formed with the Presentation Layer in which we would send subtitle and language settings to the GUI for proper display.

In terms of impacted directories and files, a majority of these changes would be found inside the Player Core Module or within GUI Settings. Some examples include:

- GUIInfoManager.cpp	- Refers to GUI Language/Subtitle Settings
- LangInfo.h	- Refers to Language Information
- \cores\VideoPlayer\	- Refers to VideoPlayer Subtitle Settings
- \cores\AudioPlayer\	- Refers to AudioPlayer Settings
- \utils\	- Language Based Utilities
- \video\dialogs\	- Refers to GUI Dialog Subtitles
- MusicInfoTag.h	- Refers to Music Information (Artists, Lyrics, etc.)
- \interfaces\	- API-related Information

4. Derivation Process

Coming up with different implementation approaches required brainstorming and research on the different ways something like this can or has been implemented. Many big-name companies like Google, Amazon, and IBM have already developed their own ASR APIs that are available for use. These APIs have been extensively trained and are paid services based on request limits. On the other hand, we could try to emulate what has already been done by these companies by creating a custom model similar to those that already exist. This would require an abundance of resources and time but would allow for the developers of Kodi to do it in a way that they like.

5. Approaches / Architectures Considered

After deliberation between our group and consideration of the multiple ways that we could explore implementing real-time STT within the Kodi platform, we decided on the two most viable approaches. This section outlines these possible approaches for the implementation of our enhancement.

5.1 Approach 1: Localized Implementation Using Deep Learning Models

In this approach, real-time AI-generated subtitles are achieved by implementing a localized speech-to-text (STT) engine to call within the Kodi platform. Instead of relying on external cloud services and third-party APIs, we can train our own deep-learning models designed specifically for speech-to-text tasks. The resulting localized engine will be equipped with custom language models, allowing for better adaptations to accents and linguistic nuances to ensure a fine-tuned transcription. This approach prioritizes self-sufficiency and reduces the dependency on external services, providing more control over the process of creating text and minimizing delay in real-time subtitle generation.

To implement this approach within the Kodi architecture, we would first need to develop and train our ML model. This will likely be a very time-consuming and extensive process that would most likely be in beta development for the foreseeable future due to the nature of what the model is expected to do. Changes would have to be made to the Data Layer to account for the massive amount of training data that must be held, and we could leverage the existing network communication protocols for *cURL* capabilities if needed. The Business Layer would house our new engine, where it would be in close contact with the PlayerCore Module upon use. The engine would function in a sort of pipe-and-filter way, receiving an input of audio, transcribing the audio, and then sending it back to the PlayerCore in ordered steps. New components would also need to be added to handle training, deployment, and utilization of the engine, as well as modifications to the various media players for communication with the engine and their subtitle-related functions. Additionally, when taking a look at the current Kodi code, there are existing parameters in place that return subtitle language information, however since our enhancement cannot guarantee a static output language, changes would have to be made to files like *GUIInfoManager.cpp* and the *dialogs* directory to account for this.

5.2 Approach 2: Cloud-Based Implementation Using Third-Party ASR Systems

In the second approach, a cloud-based implementation using a third-party API or ASR system can be used and integrated seamlessly to enable real-time STT within the Kodi platform. This approach leverages the use of a system that has already been developed to recognize and transcribe speech from extensive datasets of multilingual data sourced from the web. Because of the vast and diverse training that these models have already undergone, they are more resilient to accents, ambient noise, and specialized technical language. The use of existing APIs or systems allows for a relatively fast turnaround for implementation to enhance the overall quality of the generated subtitles and ensures scalability and reliability due to the cloud-based nature of the service.

To integrate a cloud-based ASR system into the Kodi architecture, adjustments and additions to the architecture are required. As mentioned in the previous approach, similar changes would also occur within the subtitle and language-related files within the PlayerCore Module for subtitle display and generation. In this implementation, no major changes would occur within the Data Layer due to there being no need for training or new storage, with the only major change occurring within the Business Layer. Instead of the creation of a custom engine, we would need to integrate the interface for the third-party API. This would involve communication across the network to invoke the cloud service to manage the real-time data flow for text generation.

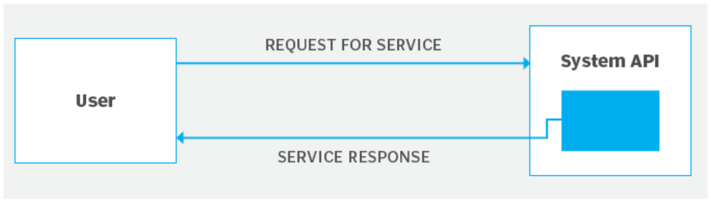

Figure 1: Diagram of how API works with a system.

The diagram above highlights the main process associated with this approach. The User (Kodi) would call the third-party API to request subtitle translation. The API would translate the data received and respond with the text to be displayed within Kodi.

6. SAAM Analysis

6.1 Major Stakeholders

Kodi Foundation: Non-profit organization that oversees the development and maintenance of Kodi. It establishes the overall direction of the platform and manages legal and financial aspects [1].

Board Members: Kodi's board members stand at the head of the Kodi Foundation and are chosen through elections held by the community. They are seen playing a role in the execution of management tasks and usually consist of former developers. These members are seen as the communicators and assessors within the community [1].

Developers and Contributors: Individuals who actively contribute to the development of Kodi by writing code, fixing bugs, adding features, and improving the overall functionality of the platform.

Add-on Developers: Developers who create add-ons and extensions for Kodi. These add-ons provide extended capabilities to the platform, consisting of additional features, and customization options.

User Community: The user base of Kodi. Their feedback, feature requests, and support all contribute to the overall growth and improvement of the platform.

Content Providers: Entities that provide legal and authorized content to the Kodi platform, such as streaming services, broadcasters, and media companies.

Hardware Manufacturers: Companies that produce devices compatible with Kodi, such as media streaming boxes, smart TVs, and other hardware.

6.2 NFRs

Stakeholders	**NFRs**
Kodi Foundation	Performance, Reliability, Maintainability, Affordability
Board Members	Compliance, Strategic Alignment
Developers and Contributors	Documentation, Modularity, Testing
Add-Ons Developers	Documentation, Compatibility
User Community	Usability, Accessibility, Customization

Content Providers	Content Protection, Broad Language Support
Hardware Manufacturers	Resource Efficiency, Compatibility

6.3 Comparing Approaches

Approach	***Pros***	**Cons**
Localized Implementation Using Deep Learning Models	**Self Sufficiency:** The localized implementation reduces dependency on external cloud services, providing more control over the process. **Reduced Latency:** Due to the processing done locally, there is potential for reduced latency for real-time subtitle generation compared to relying on external services **Privacy:** Ensures user and content data remains within the local system, mitigating potential privacy concerns with cloud-based services **Extended Use:** A localized approach can open new opportunities within Kodi, and supports reuse and expansion	**Resource Intensive:** Training and running deep learning models locally can be resource-intensive and require a significant amount of computing power and storage. **Scalability:** Scaling the system to support a growing user base may be more challenging when compared to the cloud-based approach as it relies on internal resources. **Development Cost:** Developing such a model would be complex, time-consuming, and very expensive as it requires a lot of expertise and ongoing efforts. **Constant Maintenance:** Maintaining a localized model would be an ongoing task
Cloud-Based Implementation Using ASR Systems	**Scalability:** Cloud-based services can offer high scalability, allowing for the system to handle a significant amount of users. **Diverse Training Data:** Leveraging the third-party ASR system provides access	**Dependency:** The approach relies on an external service, making the Kodi platform dependent on the availability and performance of the third-party ASR system. **Cost:** Cloud-based services involve ongoing costs.

	to the already existing extensive dataset, therefore enhancing the model’s ability to handle various accents, languages, and nuances. **Faster Implementation:** Integrating with an existing API for the ASR system allows for a much faster implementation by reducing development time.	Depending on the usage volume, this could be a significant financial expense to the platform. **Privacy Concerns:** Using external services can create potential privacy concerns, as user and content data may be processed on external servers. **Latency:** The dependency on external services may cause latency if there are delays in communication with the cloud.

6.4 Chosen Approach

The localized implementation was determined to be an overall better approach to the enhancement of the system for several reasons. First, we will look at the impacts on stakeholders. For the Kodi Foundation, this approach ensures reliability by minimizing the dependency on external services, aligning with the foundation's goal of stability. Board members benefit from compliance with privacy standards that otherwise may have been compromised by using an external API. Developers and contributors will find value in the comprehensive documentation, modularity, and robust testing processes that will occur as a result of the localized implementation of the STT engine within the system. The user community gains improved usability and accessibility with the new feature of real-time AI-generated subtitles. Content providers have their needs met with the protection of content data with a localized system, and additionally gain broad language support to their already existing content. Regarding hardware manufacturers, the compatibility of the system would remain the same, however, the resource efficiency would potentially decline as AI systems require significant resources. Despite this, our rationale was that the previously mentioned benefits outnumbered this fault. Another significant factor in the decision was the overall cost of the implementation and maintenance. The localized implementation approach is expected to provide a more predictable cost structure over time compared to the cloud-based approach, as the second approach involves ongoing expenses tied to external services and would become a significant continuous financial expense depending on the number of users that use the new feature. Additionally, the localized approach has a much higher support for future development, reuse, and customization. Although the initial cost for the development of the local ML models may be heavy, the fact that Kodi now owns all these developments as its intellectual property is massive. They can now use their property to expand or upgrade into different ML areas if needed for a much cheaper additional cost. Overall we found that using a localized approach better appealed to the priorities of each stakeholder group while advancing the overall capabilities of the Kodi platform.

7. Concurrency

Concurrency plays a significant role in the proposed enhancement of real-time generated subtitles for media. As the system concurrently processes audio input, transcribes speech into text, and updates the graphical user interface in real time, managing these tasks in parallel while maintaining efficiency is crucial for a seamless user experience. Challenges such as running resource-intensive models, latency control, and error handling must be addressed to ensure optimal performance. Some key considerations and strategies that can be implemented to help include multithreading and asynchronous I/O for capturing audio input within the Business Layer to ensure that media playback is not blocked by subtitle generation.

8. Diagrams & Use Cases

We can explore the potential real-world applications of our proposed enhancement by looking at the sequence diagrams of two specific use cases. Refer to Figure 2 to gain an understanding of the symbology being used in these diagrams.

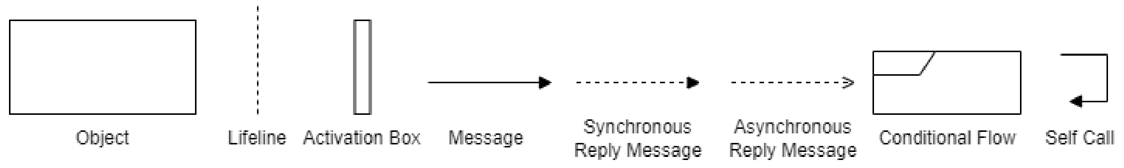

Figure 2: Sequence Diagram Legend

8.1 Use Case 1: Real-Time Subtitle Generation

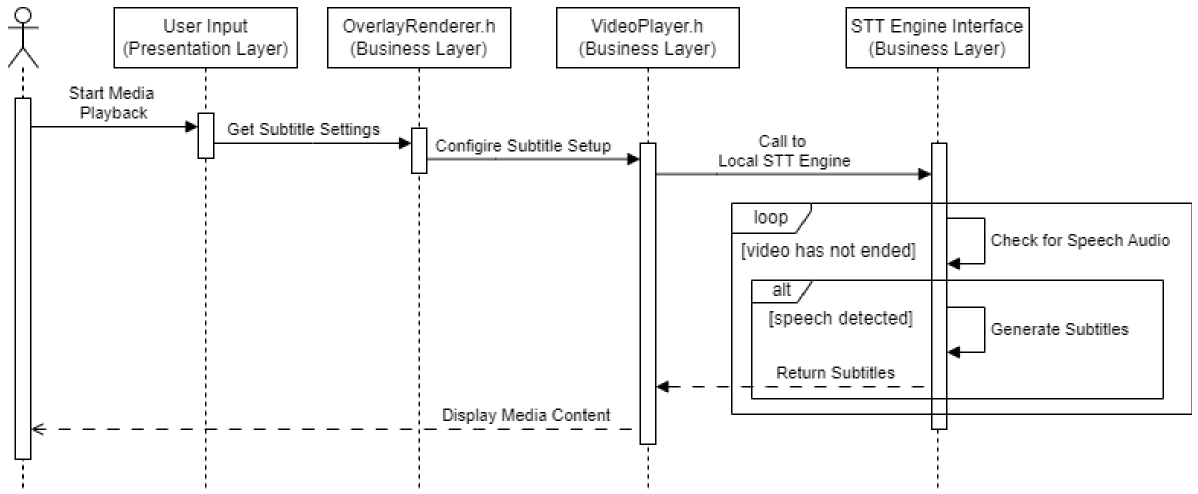

Figure 3: Sequence Diagram for Real-Time Subtitle Generation

In this use case, we will explore the process of real-time subtitle generation when watching some sort of video. For the purpose of this use case, we will assume that the process of video configuration within the system has already been set up in accordance with our concrete architecture. After starting the playback of video or even live TV, subtitle settings are retrieved from the *OverlayRenderer.h* file. This refers to location, size, and other visually based settings. Having retrieved the desired settings, the subtitle template is set within the VideoPlayer in accordance, where it would remain blank. The VideoPlayer would have a hook to the STT Engine that would be called when media playback began. Within the engine, we would have a constant listener to check for audio, and if the speech was detected the resulting text would be returned to the VideoPlayer where it would be configured and displayed using the needed subtitle-related function.

8.2 Use Case 2: Lyric Generator

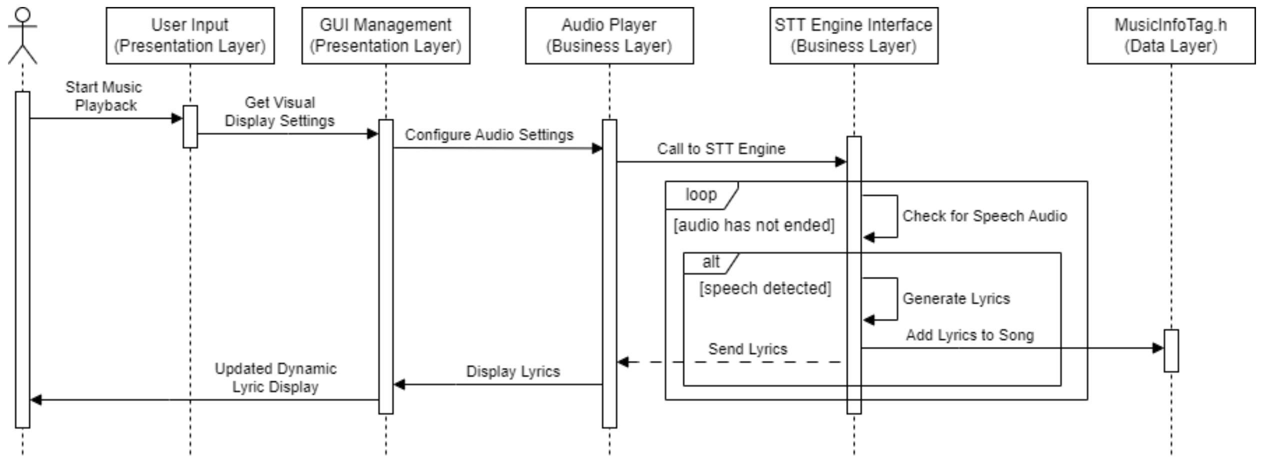
User Input (Presentation Layer)
GUI Management (Presentation Layer)
Audio Player (Business Layer)
STT Engine Interface (Business Layer)
MusicInfoTag.h (Data Layer)
Start Music Playback
Get Visual Display Settings
Configure Audio Settings
Call to STT Engine
loop
[audio has not ended]
Check for Speech Audio
alt
[speech detected]
Generate Lyrics
Add Lyrics to Song
Send Lyrics
Display Lyrics
Updated Dynamic Lyric Display

Figure 3: Sequence Diagram for Real-Time Lyric Generation

In this use case, the user is looking to use the real-time STT feature to generate lyrics for a particular song. For this use case, we will again assume the audio file has been retrieved from the Data Layer and the associated prerequisites have been taken. To begin, the user initiates the playback of music through user input. This then calls to the GUI to see how the music is to be visually displayed. Having set up the GUI as intended, the Audio Player calls the engine to generate real-time lyrics in a similar sense to a karaoke machine. This use case varies from the previous in that it saves the lyrics for a song. After lyric generation, they are sent back to the Audio Player which displays them in the GUI. The user can now see the lyrics change in real time as they listen along.

9. Plans For Testing

Testing is critical to ensure the functionality and reliability of enhancement. Both functional and non-functional requirements including usability and security need to be validated.

Due to the reality of using an AI model, results will never be perfect, however, they should be maximized. Within the engine itself, testing is crucial, and using an extensive and diverse dataset to do so is key. All languages, tones, and expressions must be considered to provide the user with the most accurate results. Inside the broad Kodi system, end-to-end testing is vital to confirm integration into media by looking at the communication across the individual layers. Unit testing drills that examine specific functionalities, subtitle requests and incorporation, and underlying ML models are also needed. Usability testing evaluates changes to the user interface, ensuring dialogs and displays are suitably configured for diverse platforms and languages. Security testing scrutinizes the secure handling of requests and generation processes, while performance testing measures system responsiveness under varying conditions. The need for regression and scalability testing is also critical to ensure the introduction of real-time subtitles does not adversely affect existing features, and to assess how well the system handles increased requests and demand within the Business Layer. User Acceptance Testing (UAT) engages real users to provide valuable insights into the overall usability and effectiveness of the new feature, collecting feedback to identify areas for improvement.

10. Potential Risks

The proposed localized implementation approach comes with several potential risks and concerns. Firstly, the resource-intensive nature of training and running deep learning models locally may impact system performance, partially on devices with limited computational resources. Additionally, the localized approach may face difficulties in recognizing diverse accents and languages due to a potentially limited dataset. Furthermore, developing a custom model for speech-to-text tasks demands a high level of expertise, often requiring specialized personnel and resources. The intricacies of training and optimizing the model lead to substantial upfront financial expenses. Not to mention, ongoing maintenance efforts, including updates and improvements to keep the model accurate, contribute to the overall cost of the implementation.

11. External Interfaces

Several external interfaces arise as a result of our chosen implementation. Our localized STT engine requires language-specific models as well as dialect support to be able to detect a variety of languages. Additionally, the gathering process or update of training data relies on the Data Layer's network communication interfaces to gather information from external sources.

12. Data Dictionary

- **Graphical User Interface (GUI):** A visual interface that allows users to interact with software using menus, controls and graphics
- **Automatic Speech Recognition (ASR):** Technology employed for converting spoken language into written text

- **Machine Learning (ML):** Subset of artificial intelligence that enables systems to learn and improve from experience without being explicitly programmed
- **Speech-to-Text (STT):** The process of converting spoken language into written text (synonymous with ASR)
- **Non-Functional requirements (NFR):** Criteria that define the system's operation, but not related to specific behaviors of functions
- **Stakeholder, Approach, Artifact, Major Stakeholder (SAAM):** Framework used for stakeholder analysis, involving the identification of major stakeholders, approaches, and artifacts within the context of the proposed enhancement

13. Lesson Learned

Throughout the exploration of enhancing Kodi with real-time STT generation, there are several important lessons that we have learned. Firstly, considerations of maintenance in terms of choosing between a third-party API and a custom ML model highlighted the tradeoffs between minimizing efforts and gaining flexibility. Testing in the domain of AI creates unique challenges, requiring examination of various languages, differences in tone, and linguistic nuances. Balancing the performance optimization of the system with the introduction of the new feature requires careful adjustments to the existing system. Stakeholder priorities, particularly regarding NFRs such as privacy, cost predictability, and platform performance, played a significant role in determining the chosen localized approach to implementation. The decision to prioritize cost predictability and future development potential over an already proven and ready-for-use implementation highlighted the difficult and careful strategic choices that developers and companies have to make.

14. Limitations

When conducting our analysis of our enhancement, we did encounter some potential limitations that may arise. Regarding the system itself, the development cost and resources associated with our chosen implementation may result in a long time for the enhancement to come to fruition. This may result in the need for the use of a paid model for feature use to recoup some of the investment cost. Additionally, limitations can occur if data is poorly sourced which may result in a poorly trained model and unsatisfactory results. Limitations can also be found in the results where there is no guarantee that the subtitles can be displayed with certain accuracy due to the nature of the task. As mentioned throughout the report, the increased overhead associated with the implementation of a local engine may also result in the feature being unavailable on certain devices or softwares. In terms of limitations of our group's understanding, none of us are experts on ML systems and what is required to implement them, however, we did our best to use our combined knowledge to estimate as closely as we could the requirements that would be needed.

15. Conclusion

The proposed enhancement of real-time AI-generated subtitles in Kodi's media player aims to improve user experience, adding accessibility for broader languages. The chosen approach is a localized one, which considers stakeholder priorities and values including cost predictability, platform performance, and self-sufficiency. Although using a third-party system may be easier to implement and maintain, the customizability and future development potential that arise from a local approach were major driving factors in our decision. The various lessons learned highlight the importance of trade-offs and stakeholder considerations in the implementation of AI-based enhancements. Overall, our proposed enhancement aims to pave the way for an abundance of artificial intelligence-based improvements in the future.

References

[1] K. Foundation, "Foundation: About," Kodi, https://kodi.tv/about/foundation/ (accessed Dec. 5, 2023).