Research Article

Enhancing Workflow Efficiency in Yocto Project: A Build Tool for Fetch Error Detection and Fixing

Huseyin KARACALI[1*], Nevzat DONUM[2*], Efecan CEBEL[3*]

[1] TTTechAuto Turkey, Software Architect, Orcid ID: https://orcid.org/0000-0002-1433-4285, E-mail: huseyin.karacali@tttech-auto.com
[2] TTTechAuto Turkey, Embedded Software Engineer, Orcid ID: https://orcid.org/0000-0002-8293-8267, E-mail: nevzat.donum@tttech-auto.com
[3] TTTechAuto Turkey, Embedded Software Engineer, Orcid ID: https://orcid.org/0000-0002-2027-0257, E-mail: efecan.cebel@tttech-auto.com
[*] Correspondence: efecan.cebel@tttech-auto.com

(First received January 18, 2024 and in final form April 12, 2024)

Reference: Karacalı, H., Donum., N., Cebel, E. Enhancing Workflow Efficiency in Yocto Project: A Build Tool for Fetch Error Detection and Fixing. The European Journal of Research and Development, 4(2), 49-76.

Abstract

Yocto Project is conceived as an initiative to provide developers with a flexible and efficient environment for preparing customized embedded Linux distributions. However, while bestowing upon developers the flexibility to create bespoke embedded Linux distributions, this project introduces various challenges. These challenges transcend mere technical proficiency, influencing critical barriers that impact the successful completion of the project. These difficulties encompass the steep learning curve of the Yocto Project, the intricacies of complex configuration files, and the management of dependencies. Developers are compelled to navigate through these intricacies within the project, concurrently encountering fetch errors arising from the continuous evolution of external sources. These fetch errors not only disrupt the flow of the project but also exhibit sensitivity to alterations in access to external resources and network-related issues. Consequently, developers find themselves expending time and effort in grappling with these challenges. The fetching process within the Yocto Project plays a pivotal role in the creation of bespoke distributions, retrieving essential external source code crucial for the development process. Fetch errors can stem from various sources, including alterations in upstream repositories and network issues, potentially hindering the progression of the project if not promptly resolved. However, the unpredictable nature of fetch errors necessitates a comprehensive solution for a seamless workflow. In this context, as a solution to these challenges, an innovative tool has been developed within the scope of this project. This tool aims to automatically detect and resolve fetch errors encountered during the preparation of custom embedded Linux distributions with the Yocto Project. The tool proficiently detects real-time internet interruptions during fetch processes and automatically initiates reattempt procedures in case of transient outages. This feature

ensures the continuous progression of the project. Additionally, the tool scrutinizes complex recipe errors within the Yocto Project, automatically rectifying issues encountered during fetch operations. This streamlined approach expedites error resolution without necessitating manual intervention from developers. The tool systematically analyzes the health of URLs employed in fetch processes, identifying potential errors in the utilized URLs. Furthermore, it evaluates internet connectivity issues arising during fetch operations within the Yocto Project. By detecting various scenarios such as DNS problems, connection timeouts, and packet loss, the tool provides developers with comprehensive reports, enabling swift diagnosis of internet connectivity issues. As a result, this tool successfully overcomes the existing challenges in resolving fetch errors within the Yocto Project. Furthermore, the tool can be extended to automatically correct not only fetch errors but also general Yocto errors. These enhancements contribute to the tool providing a more effective and versatile solution. In addition, incorporating CI/CD integration into the tool can significantly improve the quality of work. CI/CD enables automated testing and deployment of code changes, ensuring software reliability and optimizing deployment processes.

Keywords: *Yocto Project, Embedded Linux, Custom Embedded Linux Distribution, Fetch Errors, Yocto Project Learning Curve*

1. Introduction

Yocto Project is an open-source collaborative initiative hosted by the Linux Foundation. It provides developers with the capability to manage the complexity of customized embedded systems and create Linux distributions tailored to specific needs [1]. The fundamental objective of the project is to offer developers extensive configurational flexibility and enable the development of reliable, scalable, and customizable embedded Linux solutions that meet industry standards. The Yocto Project relies on foundational components such as the OpenEmbedded Build System, the BitBake compilation engine, and the OpenEmbedded-Core layers, providing users with a comprehensive toolset to establish a robust infrastructure for creating customized distributions [2]. Particularly in the automotive sector, where intricate embedded systems demand specialized solutions, the flexible and modular structure of the Yocto Project holds significant importance.

Yocto Project provides developers with extensive configuration flexibility; however, it comes with notable challenges. The learning curve of the project is steep, requiring users to invest time and effort in coping with its complexity. The learning process of Yocto Project, particularly due to the comprehensiveness of the extensive toolset, including fundamental components like the complex OpenEmbedded Build System, BitBake compilation engine, and OpenEmbedded-Core layers, is time-consuming [3]. The learning curve of the project is just one facet; developers also face challenges in technical aspects such as complex configuration files, dependency management, and fetch errors

arising from continuous data retrieval processes from external sources. These difficulties emerge as critical obstacles affecting the successful completion of the project.

Fetch errors are commonly defined as errors that arise during data retrieval processes from external sources. These errors encompass various obstacles encountered by the project when fetching data from external sources. Fetch errors are rooted in factors such as network issues, data inconsistencies arising from changes in external sources, or faulty URLs. These errors draw attention, particularly due to the project's dependency on external sources. In the development environment, where the project fetches external source codes, these errors can manifest in a broad spectrum, ranging from interruptions caused by fluctuations in network conditions to data inconsistencies stemming from changes in repository structures. In this context, the resolution of fetch errors becomes a critical element for the healthy creation of customized embedded Linux distributions. These errors significantly impact the project's workflow, necessitating developers to invest additional effort for the successful completion of the project.

The complexities of the Yocto Project, including the steep learning curve and an extensive toolset, further complicate the resolution of fetch errors. Developers, interacting with fundamental components such as the complex OpenEmbedded Build System, BitBake compilation engine, and OpenEmbedded-Core layers within the project's intricate structure, require profound technical knowledge not only for managing fetch errors but also for general system administration [3].

Developers find themselves not only dealing with fetch errors but also needing in-depth technical expertise in overall system management while navigating the intricate structure of the project, which includes components like the OpenEmbedded Build System, BitBake compilation engine, and OpenEmbedded-Core layers [3]. The complexities introduced by these factors add an additional layer of difficulty to the resolution of fetch errors.

In the context of this study, the developed tool focuses on the automatic detection and resolution of fetch errors encountered while creating custom embedded Linux distributions within the Yocto Project framework. In this regard, the tool is designed to address various challenges that may arise during the process of fetching data from external sources.

The detection of momentary internet interruptions during fetch processes is a crucial feature that forms the foundation of the tool's functionality. This enables automatic retry operations in case of temporary interruptions, ensuring the continuous progress of the project. Additionally, the tool analyzes errors in the complex recipes of the Yocto Project

and automatically corrects issues encountered during fetch operations. This allows for the swift resolution of errors without requiring manual intervention.

The tool verifies the health of the URLs used in fetch processes and detects potential errors in the URLs employed. Furthermore, it conducts a detailed analysis of internet connectivity issues that may occur during fetch operations within the Yocto Project. By identifying various scenarios such as DNS problems, connection timeouts, and packet loss, the tool provides developers with comprehensive reports.

This tool stands out as a robust solution developed to address fetch errors, aiming to optimize the Yocto Project development process. It is designed not only to tackle existing issues but also to anticipate potential situations of such errors in the future, serving as a solution to enhance the efficiency of the Yocto Project development workflow.

2. Materials

2.1. CPU

Embedded systems, designated for specific purposes, differ from general-purpose computer systems due to their optimized design for efficient operation within constrained resources. The CPUs in embedded systems are tailored to integrate features such as compact size, low power consumption, and built-in memory, prioritizing attributes like rapid response times and energy efficiency over high clock speeds [4]. Despite potentially lower clock speeds, these CPUs excel in delivering superior performance tailored to their designated tasks [5]. The variety of CPU architectures available for embedded systems significantly influences their design and functionality, with the selection of a particular architecture driven by the specific requirements and applications of the embedded system [5].

In this study, the NXP i.MX 8QuadMax CPU has been utilized as the central processing unit. The NXP i.MX 8QuadMax is a CPU designed for powerful system integration, particularly tailored for embedded applications. It features a heterogeneous multicore architecture, comprising four Arm Cortex-A72 and four Arm Cortex-A53 cores. This combination of cores is engineered to efficiently handle high-performance and energy-efficient tasks [6]. In terms of graphics processing capabilities, the NXP i.MX 8QuadMax incorporates a GC7000XSVX GPU, supporting 4K resolution videos and delivering high performance for embedded graphics applications [6]. This CPU provides an extensive range of connectivity options, including dual PCIe interfaces, dual Gigabit Ethernet, and various other data communication ports, showcasing the i.MX 8QuadMax as an

expandable and connectivity-friendly solution [6]. The security features of the NXP i.MX 8QuadMax encompass a security infrastructure that complies with industry standards, making it a preferred choice, especially in security-focused embedded applications.

The block diagram of the NXP i.MX 8QuadMax processor is illustrated in Figure 1.

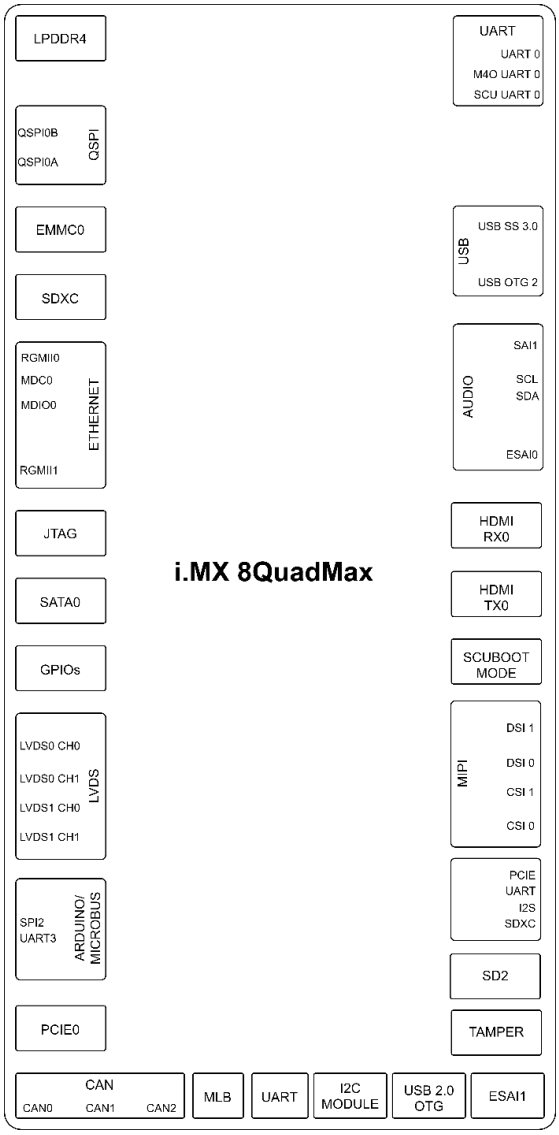

Figure 1: i.MX 8QuadMax CPU block diagram

2.2. Embedded Linux

Embedded Linux refers to an OS where the Linux kernel and associated software elements are tailored to fulfill the specific requirements of embedded systems. Utilized in embedded devices, the Linux kernel facilitates communication with hardware,

oversees processes, and manages other system resources. Typically deployed in embedded systems with specialized hardware interfaces and constrained memory and processing capabilities, Embedded Linux offers several advantages to developers [7]. As an open-source OS, Linux allows developers access to the source code for customization as needed. Supported and consistently enhanced by a large open-source community, its open nature reduces licensing expenses and facilitates seamless integration into commonly used hardware and software components.

In this investigation, the preference was for Embedded Linux as the chosen operating system. Embedded Linux boasts numerous advantages for facial recognition applications. Its robust processing capabilities ensure effective execution of intricate mathematical calculations. Extensive source code and tool support add convenience for developers, enabling swift prototyping, testing, and application development.

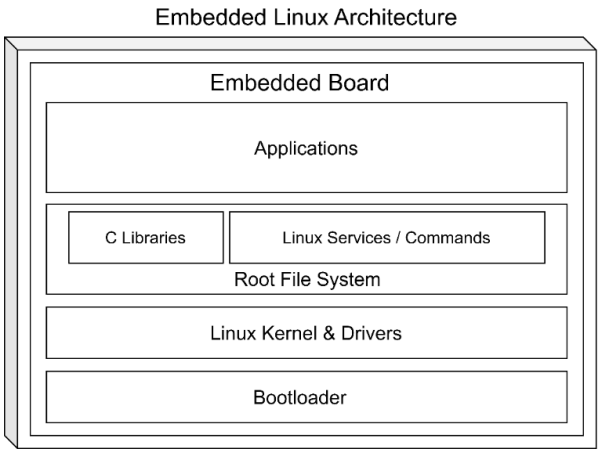

Figure 2: Embedded Linux architecture

2.3. Yocto Project

The Yocto Project is an open-source initiative providing a robust infrastructure for preparing custom embedded Linux distributions and offering developers extensive configurational flexibility [1]. The foundational element of this project is its layer structure, which presents developers with the tools necessary to build intricate and customizable Linux solutions. This layer structure amalgamates various components of the Yocto Project, adopting a modular approach that ensures flexibility and extensibility. The fundamental milestones of the Yocto Project are illustrated in Figure 3.

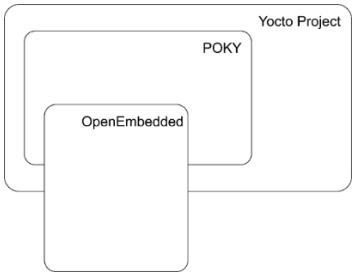

Figure 3: Fundamentals milestones of Yocto Project

The foundational layers of the Yocto Project, known as OpenEmbedded-Core layers, constitute a series of layers encompassing general system components [8]. These layers provide essential building blocks and offer developers a modular toolkit containing packaging systems, build environments, and fundamental functionality.

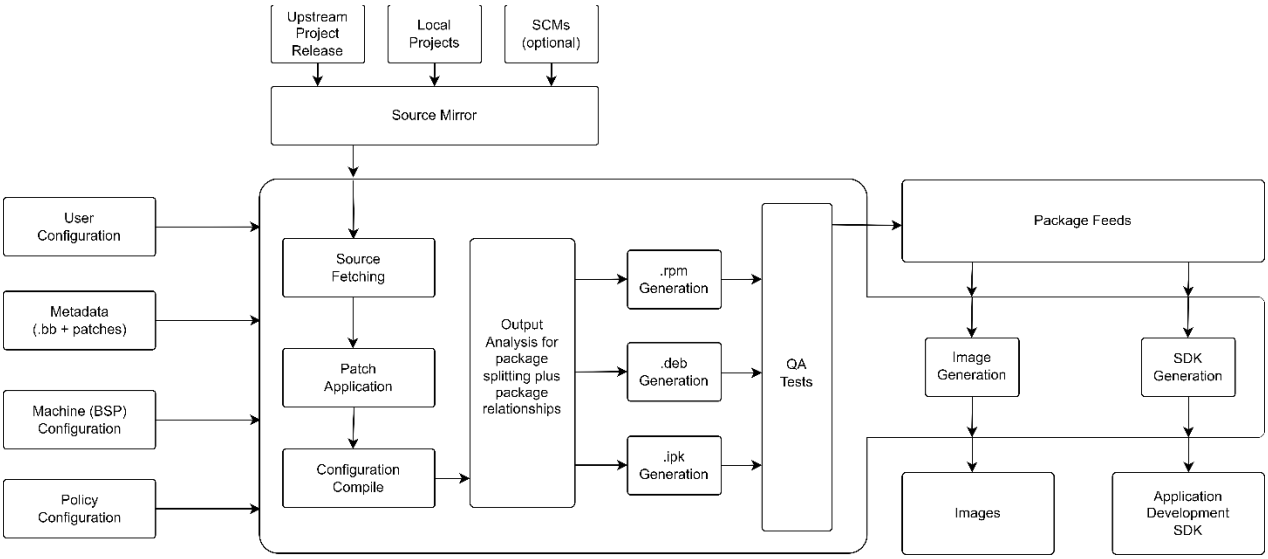

Figure 4: OpenEmbedded arhitecture workflow

The layer structure of the Yocto Project interacts with the potent BitBake compilation engine. BitBake empowers developers with capabilities such as providing compilation instructions, managing dependencies, and integrating components [9]. Recipes used in the Yocto Project define how a package should be compiled and integrated. Organized within the layer structure, these recipes grant developers the flexibility to define the necessary steps to customize their systems [9].

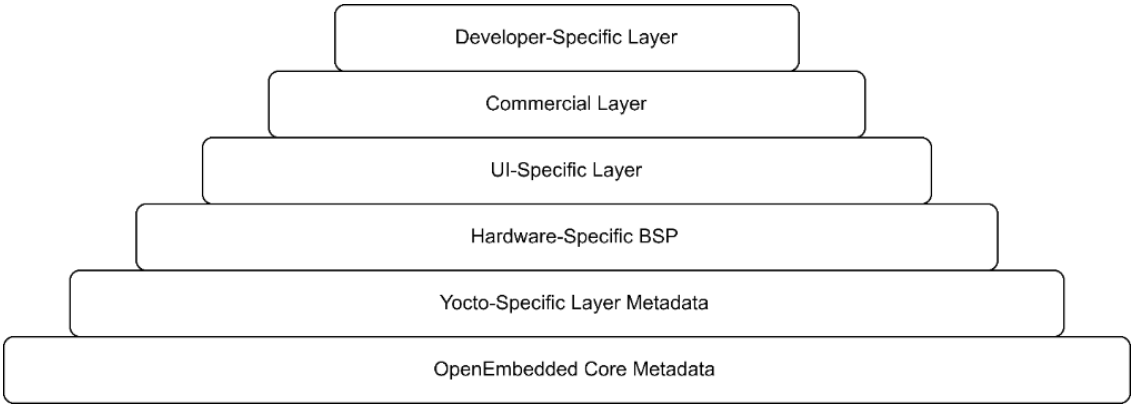

Figure 5: Yocto Project layered framework

Poky, the core distribution of the Yocto Project, holds a significant position within the layer structure [10]. Initially serving as a functional Linux distribution, Poky allows developers to initiate their journey with the Yocto Project swiftly [10]. This distribution includes a set of pre-configured recipes, providing developers with a foundational platform that can be quickly integrated into their projects.

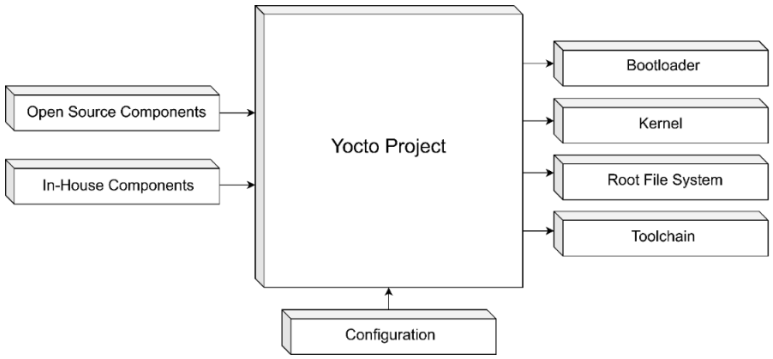

Figure 6: Yocto Project basic working diagram

The Yocto Project's layer structure permits developers to create their own layers and customize existing ones [11]. This capability enables developers to add functionality specific to their projects and take full advantage of Yocto's extensible framework.

The layer structure not only forms the core of the Yocto Project but also provides developers with the ability to customize and extend functionalities. This framework offers a powerful and flexible foundation for developers aiming to create bespoke embedded Linux solutions.

2.4. Qt Creator

A powerful integrated development environment (IDE) known as "Qt Creator" has been employed as a significant tool for software developers in this study. Developed by The Qt Company, this tool serves officially as an IDE for the Qt application framework [12]. Qt Creator is specially designed to facilitate the development, debugging, and deployment of Qt-based applications, offering a comprehensive set of features.

The IDE provides a user-friendly environment for coding in various programming languages, including C++, QML (Qt Meta-Object Language), and Python. It incorporates advanced code editing features such as syntax highlighting, auto-completion, and context-awareness, enhancing the efficiency and accuracy of the coding process [13]. Equipped with a robust debugger, Qt Creator simplifies the identification and resolution of software errors.

The integrated Qt Quick Designer streamlines the development of QML and Python-based applications, providing a visual approach to design and prototyping. Supporting version control systems like Git, the IDE enables effective code management and collaboration-focused development [14]. Its extensibility through plugins allows developers to enhance functionality according to project requirements.

In conclusion, Qt Creator stands out as a powerful and flexible IDE, providing developers with the necessary tools for seamless development, debugging, and deployment processes of Qt-based applications. Its integration with the Qt framework and support for various programming languages make it a valuable asset in the software development process.

2.5. PyQt 6

PyQt serves as a robust framework for constructing desktop applications, rooted in the Python programming language. Functioning as a set of Python bindings for Qt, a widely employed cross-platform application and UI development framework, PyQt seamlessly melds the simplicity of Python with the potent features of Qt. This amalgamation provides developers with a flexible toolkit to craft intricate graphical user interfaces. Given PyQt's nature as a Python library, it aligns seamlessly with the project's utilization of Python as its primary programming language. This alignment ensures a cohesive development experience, leveraging Python's readability and adaptability [15].

The framework chosen for the development process of the tool created within the scope of this study is PyQt 6, owing to its rich features and user-friendly design.

3. Method

The tool developed within the scope of this study aims to effectively identify and resolve fetch errors encountered during the creation process of customized embedded Linux distributions with the Yocto Project. Fetch errors typically encompass issues arising when retrieving data from external sources, causing interruptions in the project's workflow. These errors, stemming from factors such as changes in upstream repositories, network issues, and the continuous evolution of external sources, can complicate the process of creating customized distributions.

The developed tool provides a solution for overcoming these challenges, targeting developers. The tool automatically detects and resolves fetch errors, enabling the reliable and consistent creation of customized Linux distributions under the Yocto Project. This not only enhances the project's continuity but also allows developers to focus their time and efforts on more productive and efficient areas, rather than grappling with such issues.

The tool goes beyond merely identifying fetch errors; it also possesses the capability to parse and analyze errors originating from various sources. This provides developers with the opportunity to intervene more rapidly and effectively in specific issues. Additionally, by monitoring fetch processes, the tool offers a detailed view to determine the origins of errors, allowing developers to understand and resolve problems more comprehensively.

In conclusion, this research demonstrates that the developed tool makes a significant contribution by effectively addressing fetch errors encountered by Yocto Project users during the creation of customized Linux distributions.

The developed tool first checks the internet connection when the operation starts. The primary purpose of this step is to determine the presence of an internet connection for the tool to fetch data from external sources and maintain its functionality. Initially, to perform this check, the tool attempts a connection targeting the IP address 8.8.8.8. This IP address, provided by Google, generally represents a widely used and globally accessible Domain Name Server (DNS). The connection attempt directed to this IP address is utilized to determine whether a general internet access is available. Querying Google's DNS servers in this manner is a common practice to verify the existence of internet access. Checking the internet connection holds crucial significance for the tool to fetch data from external sources and sustain its functionality. This verification is a fundamental requirement for the tool to access up-to-date data and execute its operations seamlessly.

If the connection attempt to the IP address 8.8.8.8 is successful, indicating that internet access is established, the next step involves checking the DNS servers. DNS, Domain Name System, is a structure that translates domain names on the internet into IP addresses. This check aims to ensure the accuracy and integrity of data obtained from external sources by providing access to correct and reliable DNS servers. The verification of DNS servers holds critical importance, especially for resolving the URLs contained in recipes within the Yocto Project. When fetching data from external sources to create custom embedded Linux distributions, Yocto Project often retrieves this data through URLs specified within recipes. These URLs specify the correct versions, sources, or dependencies of packages and components. DNS servers are responsible for resolving the IP address associated with a domain name in the network. In other words, resolving a URL specified in a recipe involves the process of finding the IP address connected to that URL. Therefore, the availability of correct and reliable DNS servers is crucial for recipes to retrieve accurate and up-to-date information when fetching data from external sources.

If the developed tool detects any issues during the checks for internet connection and DNS servers, it generates a PDF report to notify the developer about the identified problems. The generated PDF report includes detailed information, encompassing possible internet connection issues or any problems related to DNS servers. This approach assists developers in swiftly identifying and resolving issues, ensuring that necessary corrections for the effective operation of the tool can be easily implemented.

The flowchart illustrating the Internet connection and DNS server check is presented in Figure 7.

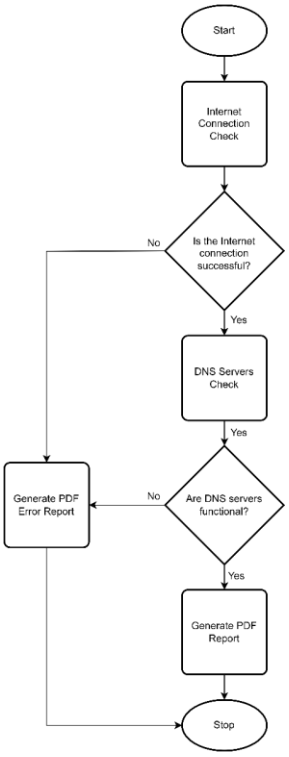

Figure 7: Internet connection DNS server check flowchart

After completing the internet connection and DNS server checks, the tool proceeds to verify the sources folder. This verification aims to ensure that all required source files for the project are complete and accurate. If the "sources" folder is not found, the tool reports this situation to the developer. This step is crucial to maintain the integrity of the project's source files and, consequently, to create a correct Linux distribution. The sources folder within the Yocto Project is a critical directory containing all external source files used by the project. This folder serves as a central location where all packages, components, and source files necessary for creating a custom embedded Linux distribution are stored. Yocto Project utilizes this "sources" folder to configure, compile, and build a custom Linux distribution.

Sources directory within the Yocto Project contains Bitbake recipe files for various components used to create customized Linux distributions. These files, denoted with the ".bb" extension, encompass comprehensive recipes that manage the processes of compiling, configuring, and installing packages. The ".bb" files contain a metalanguage interpreted by Bitbake, Yocto's package management system.

Each Bitbake recipe file provides extensive information, ranging from the method of downloading source code for a package to compilation steps, determination of dependencies, and application of configuration settings. These files enable the project to recognize and integrate packages into the system. Furthermore, Bitbake recipe files include crucial metadata such as a package's license information, version number, and repository address from which the source code is obtained. This ensures the license compliance and sustainability of the created Linux distribution.

The complexity of Bitbake recipe files varies depending on the specific requirements of each package and the features targeted by the project. These files play a central role in the creation of a customized Linux distribution, facilitating the proper integration of packages and the configuration of the system according to the desired specifications.

Sources directory serves as a repository for patch files utilized to apply customizations to the source code of packages used in the project. These patch files embody alterations such as modifying specific configurations, rectifying errors, or introducing new features to the source code. Typically denoted with the ".patch" extension, these files are applied by Bitbake recipe files during specific compilation steps.

This directory also houses the original source code files for each component used in the project. These files represent the primary source codes provided by package developers and are utilized by Bitbake during the compilation process.

Within the sources directory of the Yocto Project, configuration files used for compiling and configuring packages are present. These files dictate which features will be enabled or disabled during the compilation process of a specific package. Additionally, they contain critical settings that determine how a package will interact with other components in the system.

In this context, the presence of the sources folder in the Yocto Project workspace has been examined to verify its existence. This process has been executed utilizing the "os" module in Python, which provides access to file system and operating system functionalities. The initial step of the function involves determining the full path of the Yocto Project workspace, which is stored in a variable. Subsequently, the "os.path.join()" function is employed to create the full path of the "sources" folder by combining the Yocto Project workspace's full path with the "source" folder. This results in the generation of a path that facilitates access to the sources folder within the working directory.
Following this, the existence of the "sources" folder within the created path is verified using the "os.path.exists()" function. If the sources folder is present, the "os.path.isdir()"

function is employed to confirm whether this folder is indeed a directory. If affirmed, a notification message is displayed, indicating the presence of the "sources" folder in the Yocto Project workspace. In the event of a negative outcome, a warning message is presented, indicating that the "sources" folder could not be found.

These procedures encompass the technical steps utilized to assess the status of the "sources" folder in the Yocto Project developer's working directory and report this status. Additionally, a fundamental flowchart illustrating the verification of the sources folder is visualized in Figure 8.

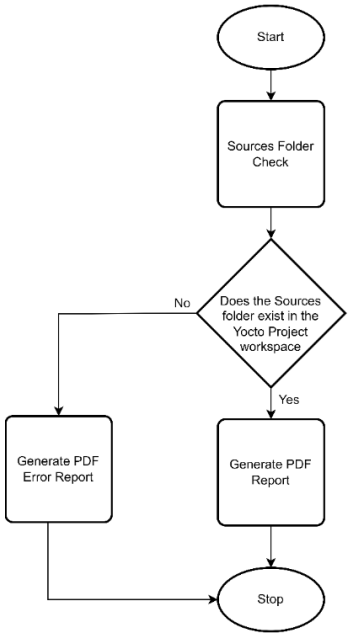

Figure 8:Verification of the sources folder flowchart

These procedures encompass the technical steps utilized to assess the status of the "sources" folder in the Yocto Project developer's working directory and report this status. Additionally, a fundamental flowchart illustrating the verification of the sources folder is visualized in Figure 8.

After the verification of the sources folder, the BitBake build engine is invoked for the build process. BitBake is a compilation and task execution tool specifically designed for embedded systems, developed for the Yocto Project. Serving as the core build engine for creating custom Linux distributions, BitBake manages tasks, dependencies, and metadata. It relies on recipes that define how packages, components, and the entire distribution should be compiled. These recipes are typically written in a specific language

and provide detailed instructions on tasks such as fetching source code, applying patches, compiling, and packaging.

Tasks represent individual steps required to compile a component, and each recipe consists of a series of tasks. BitBake ensures that these tasks run in the correct order, taking their dependencies into account. Utilizing metadata, BitBake gathers information about the compilation environment, target architecture, and package dependencies. This metadata plays a crucial role in making informed decisions during the compilation process.

The imx-image-full image has been compiled for the NXP i.MX 8QuadMax platform using the "bitbake imx-image-full" command. This image is specifically built to integrate an open-source Qt 6 framework with advanced Machine Learning capabilities. It is important to note that these images are exclusively supported for i.MX System-on-Chip (SoC) configurations equipped with hardware graphics. The utilization of Qt 6 and Machine Learning features enhances the image's capabilities, making it well-suited for applications requiring graphical interfaces and advanced computational tasks on the specified i.MX SoC hardware.

During the build process of the imx-image-full image, initiated by BitBake, a series of packages are downloaded. These download operations encompass resolving and fetching the required packages, dependencies, and components. Each package is selected to meet the intended features and functionality of the image, and it is retrieved from the relevant source repositories. Throughout this process, specific components, including Qt 6 and Machine Learning features, along with hardware graphics and other dependencies, are downloaded to fulfill the specified requirements. The download operation is automatically executed through the package management systems of the Yocto Project and BitBake's recipe files. This ensures that the imx-image-full image has all the necessary components, facilitating the creation of a embedded Linux distribution tailored to specific features.

After the build process is completed, the tool developed within the scope of this study meticulously examines the build logs. In case a fetch error is detected in the logs, the initial step involves extracting the package name causing this error from the logs. This extraction process is crucial to precisely identify the package affected by the fetch error, enabling a targeted and effective resolution process. Build logs maintain a comprehensive record of the entire build process, and the tool's analysis of these logs plays a pivotal role in ensuring the integrity and success of the build process. The parsing mechanism is

thoughtfully designed to extract relevant information related to the fetch error, facilitating subsequent debugging and resolution efforts.

When a fetch error is detected for a specific package, the process of resolving the fetch issue involves a detailed procedure. Initially, the tool determines the package name causing the error by parsing the package log file. Subsequently, the developed tool performs a reevaluation of internet connectivity and DNS server status. If no internet connection issues are identified, the procedure continues.

At this stage, the tool, for the package where the fetch error is detected, executes the "cleansstate" command within the specialized BitBake build mechanism. This command clears all temporary files in the cache of a specific package, ensuring that the package has a clean state. This step is crucial to resolving the fetch error with a clean build state.

In particular, the command "bitbake -c cleansstate <package_name>" is employed to delete all cache files related to the package with the fetch error. This action clears temporary data from previous compilation stages, allowing the package to be recompiled without being affected by previous errors. This meticulous and detailed cleanup process establishes a clean starting point for the package, serving as a preparatory step for resolving the fetch error.

In this stage, a comprehensive PDF report is generated encompassing all encountered errors during the build process. If no errors are detected in the build log file, a meticulous examination of the imx-image-full image file takes place to ensure its proper generation. The integrity and correctness of the image are thoroughly verified, and the findings are presented to the developer through a detailed PDF report.

The basic representation of this process is illustrated in the flowchart shown in Figure 9.

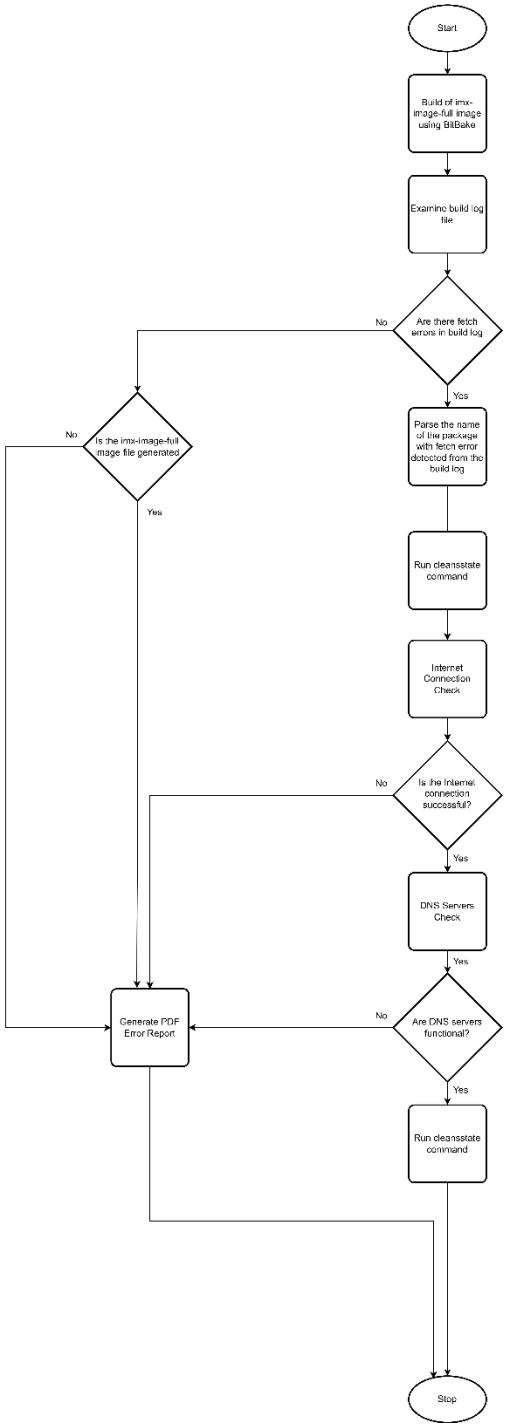

Figure 9: Detect and parse fetch error flowchart

After the removal of all cache files using the "bitbake -c cleansstate <package_name>" command for the package where the fetch error is detected, the process of resolving the

fetch error begins. In this stage, once again, the first step involves checking internet connectivity and DNS server status. In the case of no connectivity issues, the BitBake build engine is initiated, and another attempt is made to build the imx-image-full image.

Build process concludes, he generated build log file undergoes a detailed examination by the tool. The absence of any errors in the log file indicates that the tool has successfully addressed the encountered fetch error. The created image file is subjected to a thorough verification process. This verification process involves various steps to ensure the integrity and accuracy of the imx-image-full image. Upon the completion of these steps, the obtained results are presented to the developer in the form of a PDF report. This report confirms the successful completion of the build process and verifies that the generated image file aligns with the expectations.

During the examination phase of the build log file by the tool, if a fetch error is detected, and this error is not in the package where a fetch error was previously identified, the tool initiates a repetition of all these stages to rectify the fetch error. Upon identification of a fetch error during the scrutiny of the build log file by the tool, in the scenario where this error is not associated with a package previously marked with a fetch error, the tool systematically repeats all relevant stages to address and resolve the fetch error. This iterative process ensures a comprehensive and targeted approach to rectifying the specific issues identified within the build log, contributing to the overall enhancement of the build integrity.

In the examination phase of the log file, if a fetch error is detected for the same package, the tool activates an alternative method to resolve the fetch error. This alternative method relies on recipe inspection. In other words, the recipe file, which contains the compilation instructions for the package experiencing the fetch error, is thoroughly examined. The recipe file defines the compilation process and dependencies for a specific package. Recipe inspection involves a detailed examination of the instructions necessary for downloading, compiling, and integrating the package correctly. Therefore, the identification and correction of errors in the recipe file are a crucial step for resolving the fetch error. This method aims to provide a more comprehensive solution by delving into the origin of the fetch error.

Before initiating the initial build, the developer creates an environment. During the creation of this environment, the necessary host packages are installed, followed by Git configurations. The Repo tool is then installed. The repo tool manages Git repositories specified in a manifest file. The manifest file defines which repositories the project will

use, and which branches to track. This approach is particularly useful when dealing with complex structures in large projects with a series of dependencies, as it allows for the centralized control of all dependencies using a single tool. Through this tool, updating different components of the project, transitioning to different versions, or making changes in a branch becomes more streamlined. Additionally, for projects involving multiple repositories, it facilitates synchronization and update processes, ensuring integrity.

At this stage, the developer employs the Repo tool to download and synchronize Yocto Project's and NXP's provided sources.

After the developer completes the process of creating the environment and downloading the sources, when they run the tool developed within this project to initiate the build process, the tool initially backs up the base state of the sources folder under the name "backup_sources." The flow of this operation is illustrated in Figure 10 below.

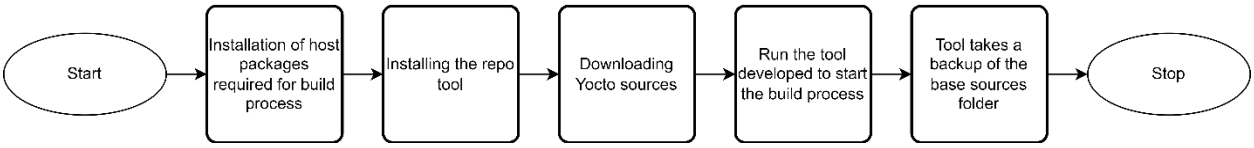

Figure 100: Prepare environment and backup sources for Yocto image build

The backup process of the base sources folder is of great significance for the recipe-based fetch error resolution method. In the case of a fetch error detected in the same package after the cleansstate process, the tool parses the SRC_URI variable within the package's recipe file. The SRC_URI variable in recipes is a crucial parameter that defines the sources specified in a Yocto recipe file. This variable specifies the source files of a component related to a recipe and typically includes a URL specified using protocols such as tar.gz, tar.bz2, git, or similar. The primary purpose of SRC_URI is to enable Yocto to fetch these source files accurately during the compilation process. This variable is used when determining the sources to be downloaded for a package, and these sources can include compressed archive files, git repositories, remote URLs, or local file paths. Yocto utilizes the information specified in SRC_URI to fetch the necessary files. The SRC_URI variable plays a significant role in the backup process, particularly in the recipe-based approach for fetch error resolution, following the cleansstate operation.

The tool parses the URL assigned to the SRC_URI variable and, upon completion of the parsing process, checks the accessibility of the obtained URL. For this verification, the requests library in Python is employed, enabling the sending and processing of HTTP

requests. Using the "requests.get" function, a GET request is sent to the parsed URL, and subsequently, the HTTP response status is examined. If the HTTP status code is 200, it signifies that an HTTP response has been successfully processed, indicating the successful completion of the client's request. This status code is particularly utilized when handling HTTP GET requests, denoting that the server has successfully located and returned the requested resources. The HTTP status codes scrutinized within the tool are comprehensively outlined in Table 1.

Table 1: Checked HTTP status codes and their descriptions

HTTP Status Code	Description
200	The request has been successfully completed.
204	The request was successful, but the server did not return any content in the response.
404	The requested resource could not be found.
500	A server-side error occurred, and the request could not be successfully processed.

The parse of the SRC_URI variable and the HTTP status query of the parse URL is basically shown in Figure 11 below.

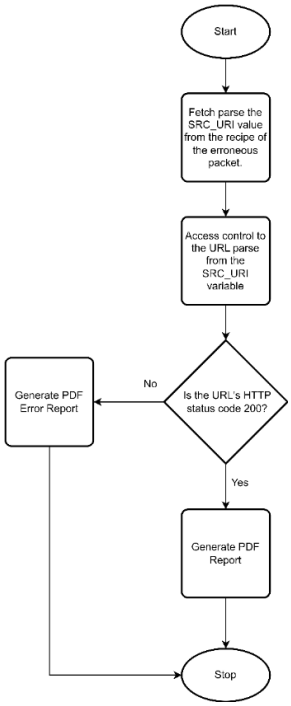

Figure 111: SCR_URI parsing and URL HTTP status code check flowchart

In case there is no error code related to the URL, the recipe file of the package in the backup sources folder is compared with the one in the current workspace. During this comparison, if any difference is detected, the base recipe is transferred from the backup sources to the workspace. After this transfer, the build process is repeated. Once the build process is completed, the build log is examined, and any errors are analyzed by the tool. If an error code related to the URL is received, the recipe file of the package in the backup sources is compared with the one in the current workspace. The SRC_URI variables of these two recipes are compared. If there is a difference in these variables, the recipe from the backup sources with the differing variable is transferred, and all build stages are repeated. If, after these steps, the fetch error has been resolved, these processes and the results are documented in a PDF report. If the fetch error persists after the steps are repeated, this situation is presented to the user in detail in the PDF report.

The general operational process and stages of the tool developed within the scope of this study are illustrated in the block diagram depicted in Figure 12.

Figure 122: General operational process and stages block diagram

4. Results

The Embedded Linux domain is standardizing around the Yocto Project as the best integration framework for creating reliable embedded Linux products. The Yocto Project efficiently reduces the time required for the development and maintenance of an embedded Linux product, enhancing its reliability and robustness by leveraging proven and tested components [16]. The Yocto Project, an open-source collaborative initiative helping developers create custom Linux-based systems, has evolved significantly over the last 12 years to meet the requirements of its community. The project continues to lead in build system technology with field advances in build reproducibility, software license management, SBOM compliance and binary artifact reuse. In an effort to support the community, The Yocto Project announced the first Long Term Support (LTS) release in October 2020. The LTS release has been extended and the lifecycle has been extended from 2 years to 4 years as standard. The support periods of the releases by date are shown in Figure 13.

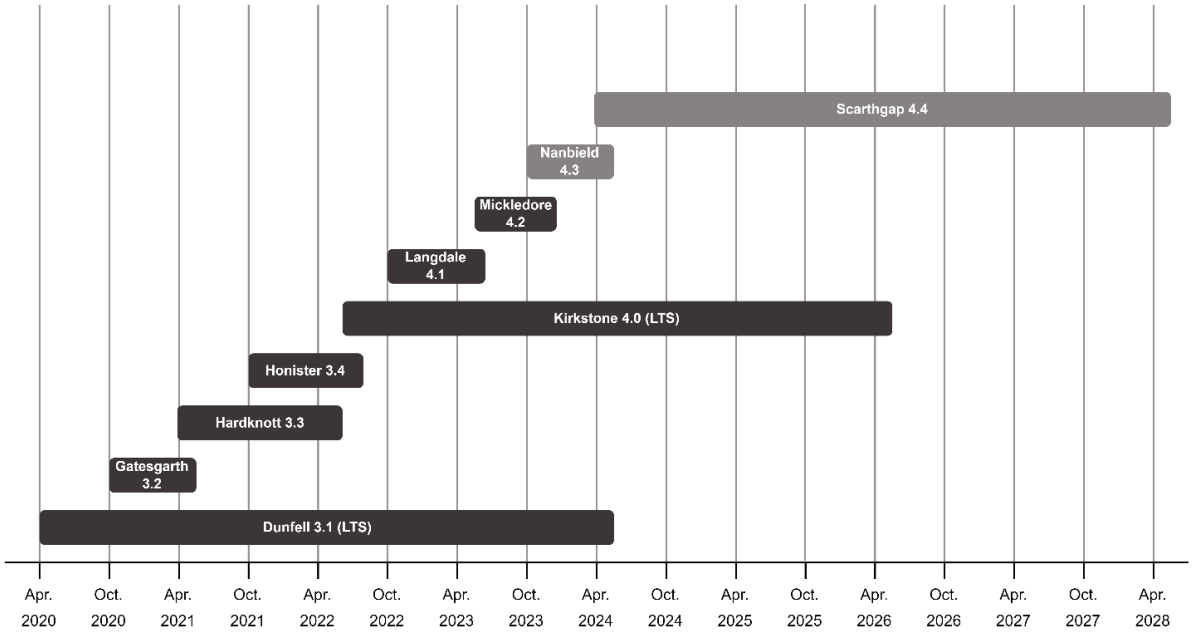

Figure 133: Yocto Project release staircase

The Yocto Project facilitates the incorporation of various architectures within a unified build tree, which encompasses multiple layers to enhance development flexibility. The layered architecture in the Yocto Project provides an organized approach to managing the filesystem and system components, enabling developers to concentrate on individual layers at their own pace. Through priority ordering, layers can be integrated, allowing higher-priority layers to supersede and modify settings. The extensive user base and community support of the Yocto Project make it convenient for developers to receive ongoing assistance from peers. Being an open-source initiative, developers have the liberty to select and customize layers without concerns about potential shifts in software vendor strategies [17].

However, despite these benefits, Yocto has a steep learning curve [18]. When issues arise in Yocto-based images, it necessitates dedicated efforts to precisely identify the root cause of the problem. Configuring settings accurately can be time-consuming. The terminology, including layers, recipes, and other elements, might initially be perplexing [19].

Yocto Project has developed GUI-based tools to shorten the learning curve and provide developers with a more effective experience. As a result of these efforts, the tools known as Hob and Toaster have emerged. Hob and Toaster are designed with a graphical user interface for the Bitbake build system within the Yocto Project [20]. Their primary goal is to streamline interaction with the Yocto Project, enabling users to perform daily tasks more quickly and efficiently. Hob and Toaster are part of initiatives aimed at optimizing interaction with projects by creating a more accessible learning curve, particularly beneficial for newcomers [21].

In addition to these efforts, the tool developed within the scope of this study aims to focus on shortening the steep learning curve of the Yocto Project and automating the resolution of fetch errors encountered in the process of creating a customized embedded Linux distribution. Thus, the goal is to go beyond the capabilities of the tools developed in this context.

The tool effectively detects internet interruptions during fetch processes and automatically initiates retry procedures in case of temporary outages, ensuring continuous project progression. Additionally, the tool thoroughly examines complex recipe errors within the Yocto Project, automatically rectifying issues encountered during fetch operations. This systematic approach aims to expedite the error resolution process rapidly without requiring manual intervention from developers. Moreover, by eliminating manual processes that developers would perform for fetch errors, the tool

prevents the BitBake lock situation and reduces the frequency of the "unable to start bitbake server" error. The tool systematically analyzes the health of URLs used in fetch processes, identifying potential errors in the utilized URLs. Furthermore, it evaluates internet connectivity issues arising during fetch operations within the Yocto Project. By detecting various scenarios such as DNS problems, connection timeouts, and packet loss, the tool provides developers with comprehensive reports, enabling swift diagnosis of internet connectivity issues.

Within the scope of this study, the visuals of the tool developed as a prototype are presented below.

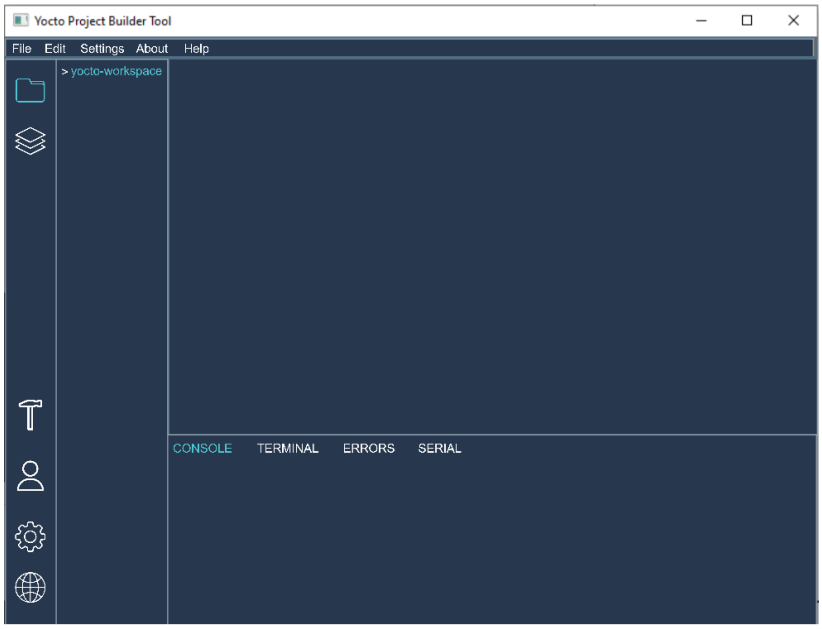

Figure 144: Main screen on Yocto Project build tool

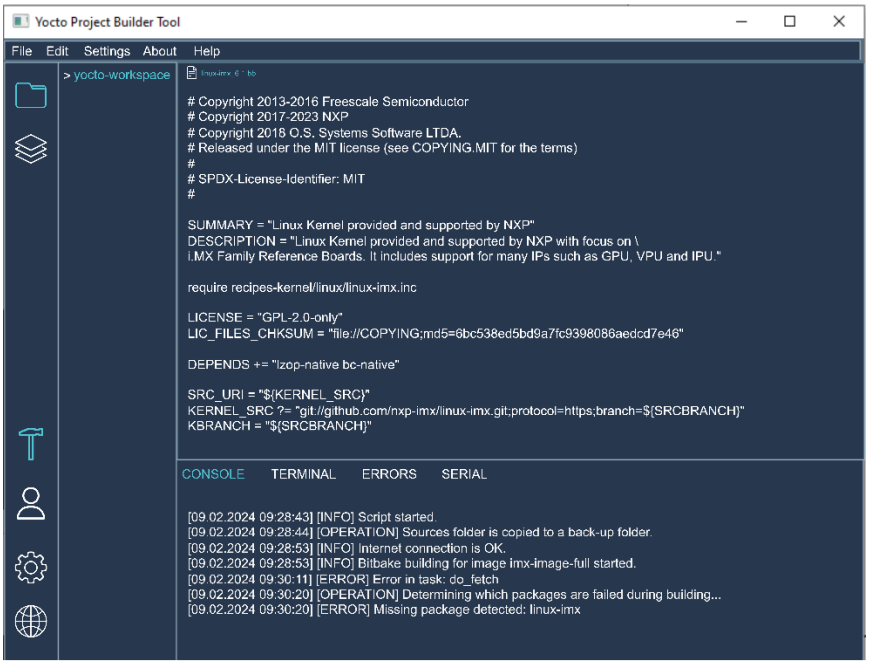

Figure 155: Build Process on Yocto Project build tool

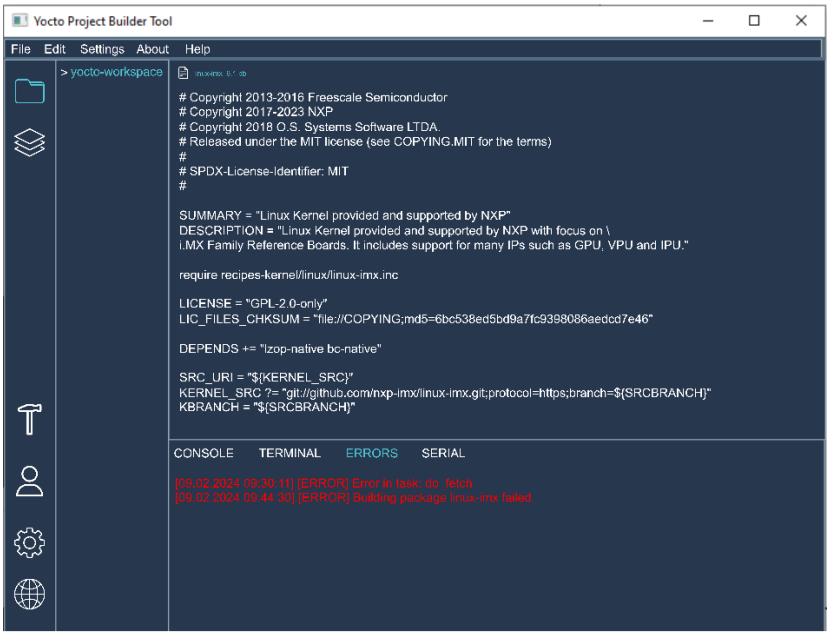

Figure 166: Error screen on Yocto Project build tool

5. Discussion and Conclusion

In the discussion section of this study, a detailed evaluation of the system's performance and applicability has been conducted. The tool developed within the scope of this study

has been successful in resolving encountered fetch errors. However, it is crucial to consider potential areas for the future development of the tool.

One of the potential areas for future development in this study is the integration of more comprehensive error detection and correction features into the tool. By automatically detecting and correcting common errors such as version incompatibilities, syntax errors, and dependency issues encountered while creating a customized embedded Linux distribution with the Yocto Project, these features added to the tool can further expedite the learning curve of the Yocto Project.

Within the scope of this study, the developed tool extends the build time to an acceptable duration due to the backup process of the necessary sources folder for recipe verification to address fetch errors. A total of 20 builds were conducted from scratch for the imx-image-full image. During these builds, backup times were measured. As a result of the tests, it was observed that the backup time for the fundamental sources folder of the imx-image-full image was completed within the range of 1.3 to 2 seconds. The tests conducted and the obtained results are depicted in the graphics below. To further advance the study, optimization of this timeframe is achievable.

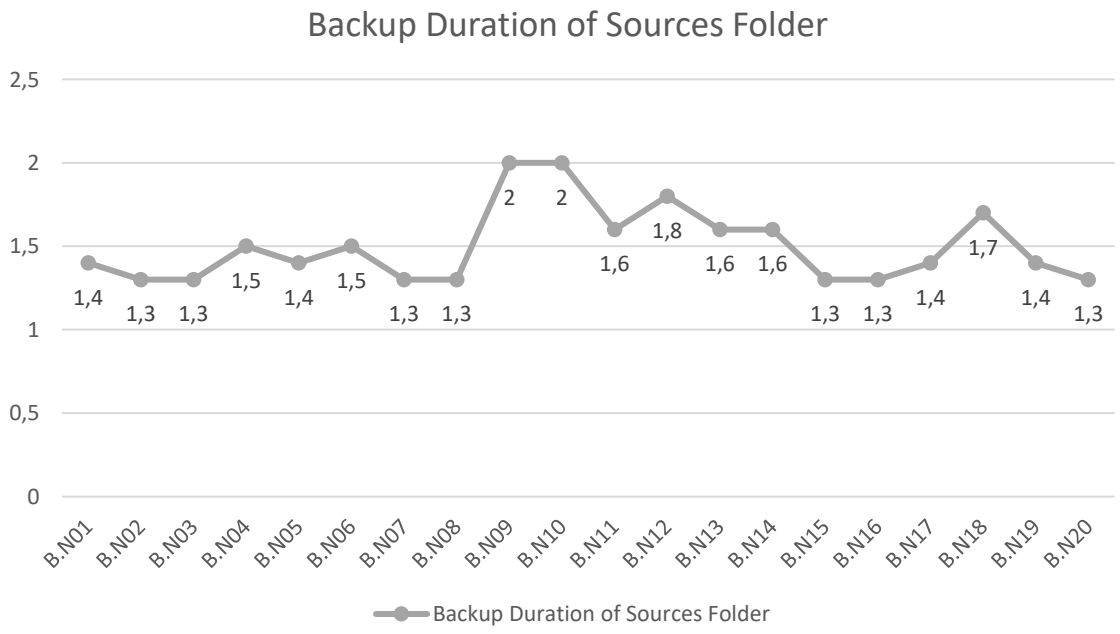

Figure 177: Backup duration of sources folder graphics

Moreover, efficiency can be enhanced by integrating a structure into the tool that automatically generates layers and recipes, which developers can utilize while creating a

custom embedded Linux distribution with the Yocto Project. By incorporating this functionality into the tools used by developers in the Yocto Project build process, a GUI-based Yocto Project ecosystem can be established, complementing the features developed within the scope of this study.

The integration of CI/CD into the tool developed in this study can enhance the quality of the work. CI/CD provides significant advantages to the Yocto Project development process. Continuous integration allows developers to automatically merge each code change and continuously test these changes. Additionally, the continuous deployment phase automatically applies successfully tested changes to target systems, thereby improving software usability and optimizing deployment processes. When used within the Yocto Project, CI/CD offers developers a faster, more reliable, and consistent development experience.

6. Acknowledge

We would like to convey our gratitude to Huseyin Karacali, the Software Architect, for his skillful guidance and motivational leadership. Additionally, we acknowledge the invaluable assistance provided by TTTech Auto Turkey throughout the development stages of the project.

References

[1] The Yocto Project, "Technical Overview - The Yocto Project," *The Yocto Project*, Nov. 09, 2023. https://www.yoctoproject.org/development/technical-overview/

[2] "2 Yocto Project Terms — The Yocto Project ® 4.3.999 documentation." https://docs.yoctoproject.org/ref-manual/terms.html

[3] "BitBake Documentation — The Yocto Project ® 4.3.999 documentation." https://docs.yoctoproject.org/bitbake.html

[4] J. Martindale, "What is a CPU? here's everything you need to know," Digital Trends, https://www.digitaltrends.com/computing/what-is-a-cpu.

[5] "Central Processing Unit," Central Processing Unit - an overview | ScienceDirect Topics, https://www.sciencedirect.com/topics/engineering/central-processing-unit.

[6] "I.MX 8 Family applications processor: ARM cortex-A53/A72/M4," i.MX 8 Family Applications Processor | Arm Cortex-A53/A72/M4 | NXP Semiconductors, https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-8-applications-processors/i-mx-8-family-arm-cortex-a53-cortex-a72-virtualization-vision-3d-graphics-4k-video:i.MX8.

[7] A. Klinger, "Embedded linux – kernel, Aufbau, toolchain," Embedded Software Engineering - Fachwissen, https://www.embedded-software-engineering.de/embedded-linux-kernel-aufbau-toolchain-a-99d15279522f4d1fcd8b2d852a8f771b/.

[8] "OpenEmbedded-Core - openembedded.org." https://www.openembedded.org/wiki/OpenEmbedded-Core
[9] "1 Overview — Bitbake dev documentation." https://docs.yoctoproject.org/bitbake/bitbake-user-manual/bitbake-user-manual-intro.html#introduction
[10] "The Architecture of Open Source Applications (Volume 2)The Yocto Project." https://aosabook.org/en/v2/yocto.html
[11] "3 Understanding and Creating Layers — The Yocto Project ® dev documentation." https://docs.yoctoproject.org/dev/dev-manual/layers.html
[12] "QT Creator Manual." https://doc.qt.io/qtcreator/
[13] "Writing Code | QT Creator Manual." https://doc.qt.io/qtcreator/creator-editor-functions.html
[14] "IDE Overview | QT Creator Manual." https://doc.qt.io/qtcreator/creator-overview.html
[15] "PyQt - Python Wiki." https://wiki.python.org/moin/PyQt.
[16] A. Gonzalez, Embedded Linux projects using Yocto Project Cookbook. 2015.
[17] D. Huong, "Development of Linux Distribution using Yocto Project," *Theseus*, 2022. https://urn.fi/URN:NBN:fi:amk-202204296507
[18] "2 Introducing the Yocto Project — The Yocto Project ® 4.3.999 documentation." https://docs.yoctoproject.org/overview-manual/yp-intro.html
[19] Admin, "Yocto Build System - Sirin Software," *Sirin Software*, Jan. 16, 2024. https://sirinsoftware.com/blog/yocto-build-system
[20] "Learning embedded Linux using the Yocto project." https://subscription.packtpub.com/book/iot-and-hardware/9781784397395/6/ch06lvl1sec42/hob-and-toaster
[21] "Toaster User Manual." https://docs.yoctoproject.org/2.1/toaster-manual/toaster-manual.html