What Slows Down FMware Development? An Empirical Study of Developer Challenges and Resolution Times

by

Zitao Wang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2025

© Zitao Wang 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Foundation Models (FMs), such as GPT-4, have revolutionized software engineering by enabling the development of *FMware* — applications and infrastructures built around these powerful models. Despite their transformative potential, FMware solutions face significant challenges in their development, deployment, and maintenance, particularly across cloud-based and on-premise platforms; this is because many of the goals, processes, tools, and technical assets of FMware development are different from those of traditional software systems. This study presents an empirical investigation of the current FMware ecosystem, focusing on three key questions: (1) what topics are most prevalent in issue discussions of FMware systems, (2) what specific challenges are commonly faced by FMware developers, and (3) what kinds of issues in FMware development have the longest resolution times? Our analysis uses data extracted from both GitHub repositories of FMware systems as well as systems hosted on popular FMware platforms such as HuggingFace, GPTStore, Ora, and Poe. Our findings reveal a strong emphasis on education, content creation, and business strategy, alongside critical technical challenges such as memory errors, dependency management, and tokenizer configurations. We further identify bug reports and core functionality issues as the most common problem types on GitHub, and show that topics concerning code review, similarity search, and prompt template design require the longest time to resolve. By uncovering insights into developer practices and pain points, this research highlights opportunities for improving FMware development tools, workflows, and community support. These insights contribute to a deeper understanding of the current FMware landscape and provide actionable recommendations for practitioners and researchers.

Acknowledgements

This thesis is based on a study currently under submission to *Empirical Software Engineering (EMSE)*, co-authored by myself, Prof. Michael W. Godfrey, and Zhimin Zhao.

I led the study and was responsible for the research design, large-scale data collection, implementation of analysis pipelines, and writing the manuscript. Zhimin Zhao assisted with collecting part of the original dataset and provided suggestions on how to process and organize the data. Prof. Michael Godfrey supervised the project, helped shape the research framing, and provided extensive feedback throughout the writing process.

I would also like to thank Prof. Meiyappan (Mei) Nagappan for his valuable advice on formulating the research direction and research questions.

Finally, I am grateful to everyone who supported me during this work, including my labmates and colleagues at the School of Computing.

Table of Contents

List of Figures **vii**

List of Tables **viii**

1 Introduction **1**

2 Background **4**

2.1 FMware Introduction and Classification 4

2.2 Challenges in FMware Development 5

2.2.1 Technical Challenges 5

2.2.2 Operational Challenges 5

2.3 State of FMware Tools and Platforms 6

2.3.1 Promptware Platforms 6

2.3.2 GitHub-Hosted FMware 6

2.4 Methodology 7

2.4.1 Goal and Research Questions 8

2.4.2 Study Design 9

3 Results **21**

3.1 RQ1 Results 21

3.2 RQ2 Results 27

3.2.1 Bug Reports Topic Modeling Results 28
3.2.2 Distribution of Challenge Topics in Bug Reports Dataset 29
3.2.3 Core Functionality Topic Modeling Results 31
3.3 RQ3 Results 33
3.3.1 Results for Bug Report Solving Time 35
3.3.2 Results for Core Function Dataset Solving Time 35

4 Discussion and Implications **38**
4.1 Implications for Researchers 38

5 Threats to Validity **40**

6 Summary **42**

References **43**

APPENDICES **46**
Appendix: Stop Word Lists Used in Topic Modeling 47

List of Figures

2.1 Study workflow to analyze the Promptware descriptions and FMware challenges. 10

3.1 Distribution of Macro-topics across different platforms 26

3.2 Challenge Taxonomy for Bug Reports Dataset Build from GitHub Issues . 29

3.3 Challenge Taxonomy for Core Functionalities Dataset Build from GitHub Issues 32

3.4 Median Solving Time by Topic for Bug Reports Dataset 34

3.5 Median Solving Time by Topic for Core Functionality Dataset 37

List of Tables

3.1 Prevalence of macro-topics and their associated topic IDs in the dataset . . 22
3.2 Issue classification distribution 27

Chapter 1

Introduction

The use of Foundation Models (FMs), such as GPT-4o and Llama, is significantly transforming many knowledge-based industries, offering capabilities that range from personalizing content recommendations on entertainment platforms to enabling predictive diagnostics in healthcare and optimizing traffic flow in urban planning. These advancements have fueled the rapid integration of FMs into diverse software applications, leading to the emergence of *FMware* — software applications and infrastructures that are built around these powerful models [15]. However, while the potential impact of FMware might seem immense, effective strategies for the practical implementation and maintenance of FMware solutions are less clear and pose numerous challenges.

FMware development involves organizing, deploying, and maintaining complex models; this often requires managing dynamic environments, handling large volumes of data, and adapting to evolving requirements. Both cloud-based and on-premise FMware solutions play an important role in this landscape. Cloud-based FMware platforms, such as GPT-store, HuggingFace, Poe, and Ora, provide scalable environments that facilitate FM deployment and management, enabling developers to leverage powerful FMs without the need to build from scratch. Conversely, on-premise FMware is often hosted on platforms such as GitHub, which offers developers greater control and customization over their FM implementations, allowing them to more easily adapt the models to their specific needs.

Despite the growing adoption of FMware, practitioners face significant challenges in managing, maintaining, and optimizing these systems. Previous research has highlighted issues such as the instability of FM prompts, the occurrence of hallucinations in generated content, and escalating costs associated with deploying and maintaining large-scale models [9, 5]. Moreover, practitioners struggle with ensuring interoperability, managing depen-

dencies, and maintaining consistency across various FMware tools and platforms. These challenges underscore the need for comprehensive empirical studies that investigate the key issues that FMware practitioners encounter, to support more effective development and maintenance practices.

Currently, practitioners face considerable challenges in working with FMware. Managing large models, ensuring stability, handling diverse requirements, and maintaining these systems over time requires significant effort [5]. Available tools and platforms do not always meet the specialized needs of FMware developers, leading to issues with managing dependencies, handling errors, and keeping models updated.

This study aims to investigate this gap by conducting an in-depth examination of FMware topics and challenges faced by developers across both cloud-based and on-premise platforms. Specifically, we seek to better understand the prevalent topics discussed in FMware communities, the core challenges practitioners face, and the time and effort required to resolve issues associated with FMware repositories. By systematically analyzing developer discussions and GitHub issues, our research aims to provide insights into the FMware ecosystem, helping to illuminate the pathways for a more efficient FMware community.

To gain a deeper understanding of the FMware landscape, we analyzed both FMware descriptions from cloud-based platforms (*Promptware*) and issue reports from FMware GitHub repositories. Our study investigates the key application areas of FMware (RQ1), the most pressing challenges developers face (RQ2), and the issues that require the most effort to resolve (RQ3):

- **RQ1: What are the most common application areas of FMware?**

  To answer this question, we analyzed 25,790 FMware descriptions collected from four cloud-based platforms: GPTStore, HuggingFace, Ora, and Poe. We applied topic modeling to extract recurring themes and organized them into a structured taxonomy of 33 fine-grained topics and 18 macro-topics.

  We found that the most prevalent macro-topics included "Educational Content & Learning" (17.7%), "Image & Visual Generation" (13.8%), and "Entertainment & Gaming" (12.7%), showing strong user interest in educational tools, visual creativity, and interactive applications. Furthermore, the presence of "Business & Strategy" (10.8%) and "Content Creation & Writing" (11.8%) also reflects broad adoption of FMware across professional domains.

- **RQ2: What challenges do developers face when building FMware?**

We examined 138,989 GitHub issues from 669 FMware repositories and automatically classified them into 15 categories. We further applied topic modeling on the two largest categories — "Bug Reports" and "Core Functionality" — to identify technical pain points.

We found that the most frequently reported issues were "Bug Reports" (30.7%) and "Core Functionality" (21.9%), suggesting that FMware development remains technically complex and error-prone. Common challenges include memory management, dependency conflicts, tokenizer context length, and model loading failures, highlighting the need for better tooling and debugging support in FMware development.

- **RQ3: Which FMware issues require the most time to resolve?**

  We focused on 30,000+ resolved issues across the "Bug Reports" and "Core Functionality" categories and measured their resolution times. We then aggregated these by topic to understand which types of issues demand the most developer effort.

  We found that "Code Contribution and Review Processes" (934.2 hours), "Similarity Search and Vector Stores" (787.5 hours), and "Prompt Templates for Model Interactions" (654.8 hours) are the most time-consuming. These results suggest that collaborative workflows, retrieval-based architecture, and user-facing prompt design are key bottlenecks in FMware engineering.

Chapter 2

Background

In this section, we provide an overview of FMware, the latest generation of software powered by Foundation Models (FMs). FMware can be broadly classified into two categories: *Promptware*, which relies on natural language prompts to generate outputs, and *Agentware*, which involves autonomous agents capable of decision making and task execution. This distinction marks a significant shift from traditional software development practices, where code-based logic designed by humans was the primary method of building systems. We examine the unique challenges posed by FMware, including handling the complexities of prompt engineering, model alignment, and managing system behaviors across these categories. Finally, we explore the tools and platforms available for FMware development, while highlighting the operational and reliability challenges in this rapidly evolving field.

2.1 FMware Introduction and Classification

Over the past decade, software engineering has undergone several transformative shifts. Traditional systems, sometimes referred to as *Codeware*, are built through manually-written logic expressed in conventional programming languages. The rise of machine learning introduced *Neuralware*, where the core development assets became data and AI models [10]. Most recently, the emergence of *Promptware* has enabled users to create applications using natural language instructions, reducing the technical barrier for building AI-powered tools [9, 5].

In this work, we adopt the broader term *FMware* to describe software systems built around Foundation Models (FMs). FMware can be classified into two main categories: *Promptware*, where user interaction occurs via natural language prompts, and *Agentware*, which

involves autonomous agents capable of multi-step reasoning and decision-making. While Promptware refers to the application layer, FMware encompasses both interactive and autonomous components grounded in FMs.

Despite their power, FMware systems pose unique engineering challenges, including hallucinations, external resource interaction limitations, and the complexity of task orchestration [15, 9]. These issues require a rethinking of the software lifecycle, from requirements engineering to system coordination, to accommodate the cognitive and architectural demands introduced by FMs.

2.2 Challenges in FMware Development

2.2.1 Technical Challenges

The development of FMware introduces several technical challenges, particularly when dealing with the instability of prompts and the occurrence of hallucinations, where foundation models (FM) generate incorrect output with high confidence[12]. These issues are prevalent across both cloud-based and on-premise FMware platforms, creating unpredictability in development. Additionally, complex multi-step tasks often require advanced prompt engineering techniques, further complicating the development process. Developers working with cloud-based FMware platforms such as GPTstore and HuggingFace face the challenge of managing these unpredictable behaviors, while on-premise solutions, such as those hosted on GitHub, require greater customization to mitigate these issues [9, 5].

2.2.2 Operational Challenges

Operationally, managing large-scale models poses significant resource and cost challenges, particularly when aligning models with enterprise-specific data. This challenge is compounded by the need to ensure scalability and stability across cloud-based environments while maintaining flexibility and control in on-premise FMware. Practitioners often encounter difficulties in monitoring the performance and reliability of their FMware systems, as the stochastic nature of FMs complicates consistent output generation and troubleshooting during the maintenance phase [9]. Understanding these operational challenges is crucial for improving the FMware ecosystem, as highlighted by the issues frequently reported by developers on platforms such as GitHub.

2.3 State of FMware Tools and Platforms

In this section, we classify FMware into two major categories based on their deployment environments and interaction mechanisms: **Promptware Platforms** and **GitHub-Hosted FMware**.

2.3.1 Promptware Platforms

Promptware refers to FMware applications that leverage pre-trained Foundation Models (FMs) and operate through natural language prompts. These platforms, such as GPTstore and Hugging Face Spaces, provide an infrastructure where developers and users can interact with models without requiring direct modifications to their underlying architectures. Promptware platforms simplify FMware adoption by offering accessible, plug-and-play environments, enabling users to customize and deploy models without extensive technical expertise.

For example, HuggingGPT, a framework built upon Hugging Face's extensive AI model repository, integrates multiple expert models to handle multimodal AI tasks spanning language, vision, and speech. It employs large language models (LLMs) such as ChatGPT as controllers to orchestrate task execution, making it a versatile solution for integrating various AI capabilities across domains [17]. Similarly, the GPTstore serves as a centralized marketplace where developers can share, customize, and deploy GPT-based applications across diverse fields, from education to business automation [18]. These platforms significantly lower the barrier to entry for FMware development, allowing practitioners to build and refine applications without constructing models from scratch.

2.3.2 GitHub-Hosted FMware

Unlike Promptware, GitHub-hosted FMware refers to open-source software projects that provide developers with direct access to FM-based system implementations, allowing for greater customization, fine-tuning, and control over model behaviors. These repositories often contain essential components such as model configurations, API integrations, fine-tuning scripts, and deployment frameworks, making them ideal for developers who require deep modifications to suit specific application needs.

GitHub-hosted FMware is particularly valuable for enterprises and research-driven teams that need to ensure data privacy, security, and extensive model adaptability. However,

maintaining these systems introduces significant operational challenges. Developers must manually manage dependencies, optimize large models for scalability, and integrate external components, which can be computationally expensive and resource-intensive [20, 6, 9]. Running large-scale FMware solutions on local infrastructure requires substantial computing power, including high-performance GPUs, memory, and storage, particularly for tasks involving real-time processing or fine-tuning [20, 14, 7]. Enterprises must also handle software dependencies and ensure high system availability, making GitHub-hosted FMware a complex but highly flexible alternative to platform-based Promptware [20, 7, 6].

Despite the rapid growth of FMware, a significant gap remains in understanding the challenges practitioners face when working with these two distinct environments. Previous research has highlighted technical concerns such as model instability, hallucination, and the need for prompt optimization [9]. However, many of these discussions remain theoretical, focusing on the capabilities and limitations of Foundation Models rather than the real-world challenges encountered by developers using Promptware platforms or GitHub-hosted FMware solutions.

For example, Promptware developers often face issues related to scalability constraints, high computational costs, and reliance on third-party APIs. In contrast, those working with GitHub-hosted FMware must navigate software dependencies, optimize large models, and manage self-hosted infrastructures, all of which require advanced technical expertise.

Our study aims to bridge this knowledge gap by conducting an empirical analysis of the major topics and challenges faced by the FMware community. We seek to uncover not only the prevalent development and maintenance issues but also the challenges presented by FMware developers. By systematically analyzing both Promptware platforms and GitHub-hosted FMware, our research aims to provide a holistic perspective on FMware adoption, ensuring that our findings are relevant to practitioners across a wide spectrum of expertise and use cases.

2.4 Methodology

This section introduces our research goals and questions, and provides a detailed explanation of our study design.

2.4.1 Goal and Research Questions

Our research aims to gain a comprehensive understanding of the major topics of FMware and the challenges within the FMware community. As FMware continues to play a critical role in various industries, it becomes increasingly important to identify the topics that are most relevant to practitioners and to understand the challenges they face during development and maintenance. Our study focuses on two types of FMwares: cloud-based FMware (*Promptware*), such as GPT Store, HuggingFace, Poe, and Ora, and on-premise FMwares hosted on GitHub. We chose these platforms based on their popularity and active communities. By examining both cloud-based and on-premise environments, we aim to provide insights that are relevant to the entire spectrum of FMware practitioners. Understanding their experiences is important for improving the FMware ecosystem and supporting more efficient development and maintenance practices.

To address this, we developed three research questions that guide our study:

- **RQ1: Which topics are most prevalent in Promptware platforms?**

  Here, we focus on identifying the major topics discussed within cloud-based FMware platforms (Promptware platform). By applying topic modeling, we can explore which topics are most prominent across different Promptware platforms. Investigating this question helps us understand what areas developers are currently interested in, and can shed light on potential areas for improvement and innovation. Knowing which topics are prevalent can help practitioners and researchers better align their efforts with the active needs and interests of the FMware community.

- **RQ2: What are the key challenges faced by developers when working with FMware?**

  We aim to improve understanding of the specific challenges and difficulties that developers encounter when working with FMware. By analyzing issues from GitHub, we aim to uncover the real-world problems that practitioners face during development and maintenance. This is important because while RQ1 provides an overview of popular topics, RQ2 delves into the practical difficulties that developers experience, offering insights into the barriers that need to be addressed to improve the FMware ecosystem.

- **RQ3: Which development issue topics require the most solving time for developers?**

Here, we examine the average resolution time for the issues identified in RQ2. By analyzing which issues take the most time to resolve, we aim to highlight the topics that demand the most effort and resources from developers. This understanding can inform the FMware community about where support, documentation, or tooling improvements are most needed, helping to reduce the overall maintenance burden on practitioners.

By addressing these research questions, our study not only identifies the most prevalent topics within the FMware community (RQ1) but also reveals the practical challenges (RQ2) and the time-consuming issues (RQ3) that developers face. This approach provides a comprehensive view of both the current state and the difficulties of FMware development, offering insights for practitioners, researchers, and tool developers aiming to improve the FMware ecosystem.

2.4.2 Study Design

We present the workflow of our study in Figure 2.1. Our study begins with the collection of cloud-based FMware descriptions from popular closed-source platforms for RQ1. Next, we preprocess these collections of descriptions and use BERTopic technique to perform topic extraction. We then group the results into "macrotopics" and compare the different distributions across different platforms. Then, we collect issues from the FMware community on GitHub for RQ2. Based on the issue tags given by the developers, we build a series of categories for the issues and analyze the distribution of different types of issues. Then we perform topic modeling on those categories in which developers and users may have interests. Finally, we evaluate and compare the average solving time for issues and clarify what topics of issues require the most developer efforts to solve for RQ3.

Phase 1: Topic modeling for PromptWare descriptions

In this phase, our goal is to collect a set of FMware systems and their descriptions from multiple cloud-based close-source platforms. We select our dataset based on the population of PromptWares and popularity of the platforms. From this data collection, we applied topic modeling which yielded the distribution and "hot" trends in closed-source FMware.

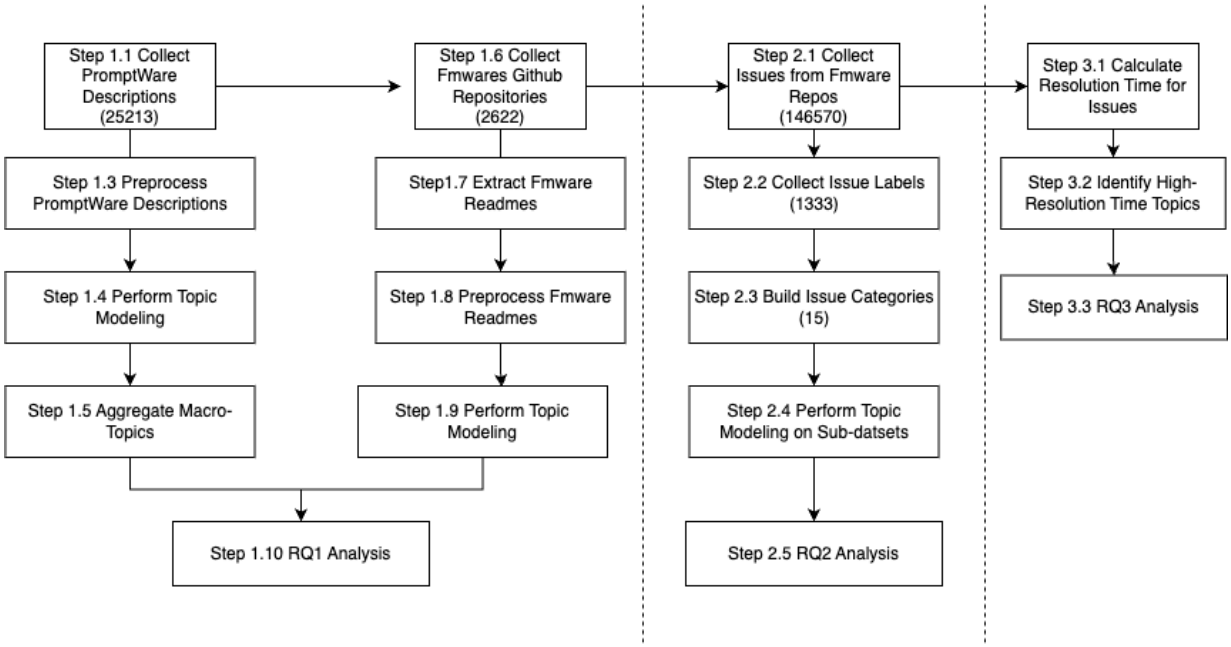


Figure 2.1: Study workflow to analyze the Promptware descriptions and FMware challenges.

Step 1.1: Collect PromptWare descriptions

We begin by reviewing a range of popular closed-source FMware platforms in the market such as GPT Store. Our goal is to identify FMware platforms that host general-purpose FMware systems, similar to how traditional app stores offer a wide range of applications. After filtering, we chose three top platforms based on size (GPT Store, POE, and Ora) and one platform based on its broad popularity and recent rapid growth (HuggingFace Assistant platform) as the basis of our dataset. In the end, we collected 90,383 app descriptions from GPT Store, 12,380 from Poe, 4,672 from Ora, and 541 from HuggingFace.

We select the data set for topic modeling to ensure a balance between representativeness and computational feasibility. The data set includes 10,000 app descriptions from both GPT Store and Poe, together with the full datasets from both Ora (4,672 app descriptions) and HuggingFace (541 app descriptions). The decision to select 10,000 apps from the two larger platforms provides a robust yet manageable sample size for capturing the diversity of apps without overwhelming our computational resources. Meanwhile, the smaller datasets from Ora and HuggingFace were included in full due to their relatively small size but likely significant value, particularly HuggingFace, which is rapidly expanding in both app count and popularity. This selection ensures that we have sufficient data for meaning-

ful topic analysis across different FMware platforms while maintaining a practical scope for the analysis.

Step 1.2: Preprocess PromptWare descriptions

After extracting PromptWare descriptions from the selected platforms, we preprocess the text using standard natural language processing (NLP) techniques:

1. Tokenize the input stream.
2. "Scrub" the tokens to eliminate non-alphabetic characters such as numbers and punctuation marks.
3. Using the `spaCy` NLP library, identify and retain only *nouns* and *verbs*, as they are the most meaningful parts of speech for our analysis [13].
4. Eliminate unnecessary words to enhance topic modeling clarity. We categorize stop words into three distinct types:
   (a) **General stop words:** Common words in English (e.g.,"you", "like", "the") that do not contribute to meaningful topic extraction.
   (b) **Platform-specific words:** Words that occur frequently in the FMware descriptions but contribute little discriminatory value, such as "GPT", "GPTStore", and "POE". We extracted the most frequent words from FMware descriptions and manually reviewed them to filter out terms that were highly repetitive yet semantically uninformative for topic modeling. Words that were excessively platform-branded (e.g., "GPTStore") or redundant in distinguishing meaningful topics were added to the stop word list.
   (c) **FMware-specific terms:** Words closely associated with FMware functionalities but not beneficial for topic modeling, such as "AI", "Bot", and "Chatbot".

   We evaluate each stop word by analyzing coherence scores and reviewing the top 10 keywords for each topic. A full list of our stop words can be found in Appendices. 6

Following these steps, the preprocessed text is ready for input into BERTopic for hyperparameter tuning and subsequent topic extraction.

Step 1.3: Perform Topic Modeling with BERTopic

We employ BERTopic[1] to perform topic modeling on the preprocessed PromptWare descriptions. BERTopic offers several advantages over traditional topic modeling approaches, making it particularly well-suited for our analysis. For example, unlike models such as bag-of-words or n-grams, which are sparse and high-dimensional, BERTopic leverages *BERT embeddings* to capture rich semantic nuances in text.

BERT (Bidirectional Encoder Representations from Transformers) embeddings are dense vector representations of text generated by a transformer-based model. Unlike traditional embeddings, BERT captures both left and right word context, improving semantic understanding. This approach has been found to enhance topic coherence, leading to more meaningful topic clusters [8].

To optimize the performance of BERTopic, we conduct comprehensive hyperparameter tuning. Guided by official BERTopic documentation and insights from its GitHub community, we experiment with parameters such as *n_grams*, *min_cluster_size*, and *n_components*. Additionally, we incorporate the top five embedding models from HuggingFace's Massive Text Embedding Benchmark (MTEB) leaderboard[2] to identify the best-suited embeddings for our dataset.

During the tuning process, we evaluate the generated topics using Coherence C_V and Coherence UMass scores [19], which are widely-adopted metrics for assessing the quality of topic models. If coherence scores improve significantly at the boundaries of the hyperparameter search space, we heuristically expand the search range for further refinement. Conversely, we halt adjustments when performance begins to degrade, ensuring stability and reliability in the final model.

In cases where coherence scores are similar across different configurations, we consider additional metrics, such as the number of topics generated and the proportion of descriptions that align with identified topics. To further ensure topic quality, we manually review a random sample comprising 5% of the clustered descriptions from each topic. This step helps validate the semantic relevance and consistency of the topics.

If topics appear unclear, overly broad, or exhibit significant overlap, we revisit earlier preprocessing steps or refine hyperparameter settings. This iterative approach ensures that the final topics are both distinct and interpretable, providing actionable insights into the thematic structure of PromptWare descriptions. By integrating robust tuning practices and manual validation, we enhance the reliability and utility of our topic modeling results.

Step 1.4: Organize into Macrotopics

[1] `https://github.com/MaartenGr/BERTopic`

[2] `https://huggingface.co/spaces/mteb/leaderboard`

After applying BERTopic, we identified a total of 33 distinct topics from the FMware descriptions. To enhance interpretability and reduce redundancy, we aggregated these topics into 18 macrotopics. This process involved grouping semantically similar topics that shared common themes or closely related subject matter. We used the "negotiated agreement" technique [2] to enhance the reliability of the coding process. This approach involved collaborative coding among the authors, ensuring consensus and refining topic definitions; similar techniques have been effectively employed in prior studies, such as those by Chen et al. [4], demonstrating its value in improving coding accuracy and consistency.

By consolidating overlapping or narrowly defined topics, the resulting macro-topics provide a higher-level and more structured representation of the dataset, making it easier to identify dominant themes and trends. This aggregation reduces fragmentation, ensures meaningful topic differentiation, and allows for a clearer analysis of the FMware ecosystem.

Step 1.5: Collect FMware Repositories

The primary motivation for collecting FMware repositories is to establish a baseline for comparing FMware (i.e., data from GitHub repositories) with PromptWare applications (i.e., data from platforms such as GPT Store, HuggingFace, POE, and Ora). FMware systems often emphasize technical depth, open-source collaboration, and community-driven development, while PromptWare systems focus on user-facing applications and general-purpose functionalities. By analyzing both, we aim to highlight the similarities and differences between these two ecosystems and reinforce the validity of our topic-modeling results. Additionally, GitHub repositories provide rich metadata, such as README files and issue discussions, which offer deeper insights into developer challenges and priorities.

To build our FMware repository dataset, we systematically searched GitHub using domain-specific tags and keywords, such as "GPT", "LLM", "FMware", and "AI tools." This approach ensured a wide coverage of repositories relevant to FMware development.

After collecting an initial set of repositories, we applied rigorous exclusion criteria to refine the dataset:

- **Inactive Repositories:** We excluded repositories that had not been updated in the last 12 months, to ensure relevance and currency.
- **Lack of Documentation:** We excluded repositories without at least minimal documentation (e.g., a README file or contributor guide), as these lack the necessary information to be able to perform our analysis.

- **Non-English Repositories:** We excluded repositories primarily written or documented in languages other than English, for consistency in analysis.
- **Non-FMware Repositories:** We excluded repositories that, after manual analysis, we felt did not pertain to FMware development or did not align with our research focus, as they were out of scope.
- **Low Engagement Repositories:** We excluded repositories with zero stars or forks, as these often indicate a lack of community interaction or development activity.

By applying these filters, we refined our dataset to 669 valid repositories.

The collected FMware repositories serve as a baseline for comparing their functionality and themes with PromptWare descriptions obtained earlier in Step 1.1. This baseline enhances the robustness of our topic modeling approach, allowing us to validate whether the topics generated from PromptWare datasets align with or diverge from those observed in the FMware domain.

Step 1.6: Create Preliminary Categories for FMware Systems

To further analyze the FMware repositories collected in Step 1.5, we extracted the README files from each repository. These README files provide detailed descriptions of the purpose, functionality, and use cases of the FMware systems, making them a rich source of information for categorization and analysis.

Given the large size of the dataset, we performed stratified random sampling to ensure representativeness while maintaining statistical rigor. Specifically, we sampled the README files at a 95% confidence level with a 5% margin of error. This process resulted in a manageable subset of README files that retained the diversity and variability of the full dataset.

To categorize the sampled README files, we employed a card-sorting methodology; card sorting is a widely used technique for organizing qualitative data into meaningful categories [2]. Each sampled README was manually reviewed, and its content was sorted into preliminary categories based on themes, keywords, and functionality described in the text.

Through iterative discussions and consensus building, we established a set of preliminary FMware categories. These categories serve two purposes:

- They provide a structured understanding of the FMware ecosystem by grouping repositories with similar functionalities or goals.

- They serve as a baseline for comparing the FMware categories with those derived from PromptWare descriptions (Step 1.5).

This categorization process helped to align our understanding of FMware functionalities with the broader goals of our study. It also provided initial insights into the alignment and divergence between FMware and PromptWare applications, reinforcing the validity of our topic modeling results.

Phase 2: Topic modeling for Github Issues from FMware repositories

In this phase, our focus shifts to gathering GitHub issues from FMware repositories to further analyze the challenges, feature requests, and overall developer experiences related to FMware. By examining these issues, we aim to gain deeper insights into the practical concerns and improvements raised by users and contributors. These insights complement the trends identified in Phase 1 by highlighting the real-world problems developers face when using FMware and how these issues align with the broader topics identified through topic modeling. This phase involves systematically collecting and categorizing issues from selected GitHub repositories to ensure comprehensive coverage of the common challenges and opportunities for enhancement within FMware ecosystems.

Step 2.1: Collect FMware repositories

To collect FMware repositories, we leveraged GitHub's "awesome lists" to identify relevant FMware projects. These curated lists are a valuable resource for discovering high-quality repositories across various domains. We browsed through all "awesome lists" on GitHub that relate to FMware, using keyword filtering to narrow down the search. Keywords such as "GPT" were employed to focus on repositories aligned with our target technologies. A full list of the search terms is provided in the replication package[21]. After filtering, we identified and compiled 2,622 FMware repositories that met our criteria for further analysis. After collecting the repositories, we apply the exclusion criteria from step 1.5 of section 2.4.2 Applying these filters to our initial candidate set of 2622 repositories results in a final set of 669 repositories.

Step 2.2: Collect Issues

In this step, we focus on collecting issues from the FMware repositories identified in Step 2.1. GitHub issues provide information on challenges, bugs, feature requests, and user feedback related to FMware development. We use a three step approach for issue collection:

- *Filtering active repositories*: We target repositories that meet the inclusion criteria described in Step 2.1.
- *Issue retrieval*: For each repository, we collect both open and closed issues. Closed issues represent resolved challenges or completed feature requests, while open issues highlight ongoing problems or discussions.
- *Issue metadata*: Along with the issue content, we gather metadata such as creation date, labels, number of comments, participants, open time, and closing time. By analyzing the open and close times, we can measure how long it takes to resolve each issue, providing insights into the efficiency and responsiveness of the development process.

Following these steps resulted in a total of 146,570 valid issues collected from the FMware repositories. These issues, along with their associated metadata and resolution times, form the foundation for our analysis of the challenges, trends, and responsiveness within FMware development.

Step 2.3: Open Card Sorting to Build Issue Categories

In this step, we use the open card sorting method to categorize the issues collected in Step 2.2. From the dataset, we identified 1,333 unique issue labels, which reflect various types of problems, feature requests, and other developer concerns. Using open card sorting, we systematically group these labels based on their similarities and thematic relationships. Through this process, we developed 15 distinct categories for issue labeling. We labeled the categories to help focus on the fields of greatest interest to our research. By organizing the issues into these 15 categories, we can more effectively target and analyze the topics most relevant to FMware development, such as bug reports, performance issues, feature requests, or documentation improvements. This structured labeling system allows us to prioritize the areas that align with our study's goals and gain deeper insights into specific challenges and trends within the FMware ecosystem.

Step 2.4: Mapping Issues to Categories Using Labels and AI-Assisted Classification

In this step, we map each issue to one of the 15 predefined categories that we found in the previous step, leveraging issue labels where available. For the 48,700 labeled issues, we directly map them to an appropriate category based on their labels. However, 98,870 issues in our dataset were unlabelled, necessitating an additional step for classification.

To categorize these unlabeled issues, we utilized the **Claude 3-Haiku model**. This AI model was used to classify the issues based on both the issue title and issue body. The model was prompted with a predefined set of 15 categories, and for each issue, it classified the content by matching it to one of these categories based on semantic analysis.

The prompt used for the classification is as follows:

```
You are a professional SE researcher and categorize issues accurately.
Classify the following Github issue titles into these topics by number:
1. Bug Reports
2. Feature Requests
3. Documentation
4. User Experience (UX)
5. Security
6. Core Functionality
7. Project Management
8. Quality Assurance & Testing
9. Infrastructure & Deployment
10. Support & Questions
11. API & Integration
12. Platform Compatibility
13. Performance & Optimization
14. Miscellaneous
15. Internationalization & Localization

For each issue, respond with the issue number and the corresponding
topic number, one per line.

Titles and Bodies:
```

This method ensured that even the issues without pre-existing labels were systematically categorized, allowing us to maintain a comprehensive and consistent dataset for further analysis. The combination of label-based and AI-assisted categorization enabled us to handle the large volume of data effectively.

Step 2.5: Preprocessing Issues for Topic Modeling

To prepare the Bug Reports and Core Functionality issues for topic modeling, we applied a similar preprocessing pipeline as described in Step 1.2 for PromptWare descriptions. We

reused the same NLP framework and processing flow, with slight adjustments (indicated by *italicized text*) tailored to GitHub issue data:

1. Tokenize the input stream.
2. "Scrub" the tokens to eliminate non-alphabetic characters such as numbers and punctuation marks.
3. Using the `spaCy` NLP library, identify and retain only *nouns* and *verbs*, as they are the most meaningful parts of speech for our analysis [13].
4. Eliminate unnecessary words to enhance topic modeling clarity. We categorize stop words into three distinct types:
   (a) **General stop words:** Common words in English (e.g., "you", "like", "the") that do not contribute to meaningful topic extraction.
   (b) **Discussion-specific words:** *Words such as "discuss", "post", and "difference" that occur frequently in GitHub issue discussions but add little value for identifying topics.*
   (c) **Preserved diagnostic terms:** *Unlike the PromptWare descriptions, we retained FMware-relevant terms such as "error", "bug", and "crash" because they carry essential diagnostic information.*

   We refined the stop word list by evaluating topic coherence scores and manually inspecting top keywords, as in Step 1.2.

As with PromptWare descriptions, we refined the stop word list through coherence score evaluation and manual review of top keywords per topic. The final preprocessed text was then input into BERTopic for modeling. For reproducibility,the complete stop word list is provided in the Appendix.6

Step 2.6: Apply Topic Modeling to Selected Categories

After preprocessing the issues in Step 2.5, we applied BERTopic to perform topic modeling on the two selected categories: Bug Reports and Core Functionality. Following the approach outlined in Step 1.5, we identified distinct topics and then aggregated them into broader macro-topics to reduce redundancy and improve interpretability. This process allowed us to group semantically similar topics, ensuring that the final set of topics was comprehensive and distinct.

We then evaluated these macro-topics based on their "prevalence", consistent with the method described in Step 1.6. Prevalence, which measures the frequency with which a topic appears in the dataset, helped us prioritize the most relevant macro-topics for deeper analysis. In addition, we identified low prevalence topics that may still hold significance in niche contexts. This evaluation provided a clearer understanding of the key themes within "Bug Reports" and "Core Functionality", guiding the next steps of our qualitative and quantitative assessments.

Phase 3: Analyzing Issue Resolution Time by Topic

In this phase, we address RQ3: Which topics of issues require more solving time for developers? Based on the topics identified in RQ2, we analyze the average resolution time for each topic to evaluate which categories demand the most developer effort.

We begin by calculating the resolution time for each issue that has been closed, which we define as the time between the issue's opening and closing. This data is then grouped by the topics generated through topic modeling in Phase 2. Our focus is on categories that developers and users are likely to prioritize, such as "Bug Reports" and "Core Functionality", as these tend to attract the most attention and have a significant impact on FMware development.

By analyzing the median and mean solving time across different topics, we aim to identify which issues require the greatest investment of time and resources from developers. This analysis will help highlight the areas where developer efforts are most concentrated and provide insights into potential bottlenecks in issue resolution. Understanding which topics take the longest to resolve will inform strategies for optimizing developer workflows and improving the overall efficiency of FMware development.

Step 3.1: Calculate Resolution Time for Each Topic

In this step, we calculate median and mean resolution time for each topic. For each issue, the resolution time is determined by taking the difference between the issue's open time and close time. This time difference reflects how long it took for the issue to be resolved. Once we have the resolution time for all issues, we group them by their assigned topics from Phase 2.

For each topic, we then compute the median and mean resolution time to identify how long issues in that category typically take to resolve. We primarily focus on the **median** as it is more robust to outliers and better represents the typical resolution time. The mean is

also reported to provide additional context where relevant.

This step will provide a clear understanding of which topics require more or less time for developers to address, helping us to identify patterns and potential bottlenecks in the development process.

Step 3.2: Identify High-Resolution Time Topics

Once the resolution time for all topics is calculated, we proceed by ranking the topics based on their median resolution time. We also examine the topics with the highest mean resolution times for comparison, offering supplementary insight into topics with highly variable or prolonged resolution durations. These high-resolution time topics likely represent complex or resource-intensive issues that may warrant further attention or process improvements.

This ranking allows us to identify which topics are more time-consuming for developers to resolve. By highlighting the topics with the highest mean resolution times, we can pinpoint areas where developers are spending the most effort and time. These high-resolution time topics likely represent complex or resource-intensive issues that may require further investigation or prioritization for process improvements. This step thus helps to focus attention on potential bottlenecks or areas that may need additional resources to reduce issue resolution times.

Chapter 3

Results

In this section, we present the results of our investigation into our three research questions.

3.1 RQ1 Results

In RQ1, we examine the distribution of macro-topics across various FMware platforms to understand the differing focuses and interests within the closed-source FMware ecosystem. We begin by categorizing FMware descriptions into 33 fine-grained topics, which are then aggregated into 18 macro-topics through manual labeling based on semantic similarity and domain knowledge. This classification helps to reveal high-level trends across the ecosystem.

Each macro-topic is accompanied by a clear definition and an illustrative example to highlight its main characteristics. All examples are direct quotes from the FMware descriptions in our dataset, selected to best represent the theme of each topic. Table 3.1 presents the relative frequency of each macro-topic across platforms. A complete list of the 33 fine-grained topics, their IDs, and representative examples is included in the Appendix for reference.

From the topic modeling and manual labeling, we identified 33 topics and combined them into 18 macro-topics.

Macro-topic	Topic IDs	Prevalence (%)
Educational Content & Learning	1, 6, 28, 30	17.74
Image & Visual Generation	0	13.78
Entertainment & Gaming	3, 12, 16, 19, 29	12.69
Content Creation & Writing	7, 8, 9, 23, 24	11.78
Business & Strategy	2, 22, 33	10.75
Technology & Programming	4	6.11
Health & Wellness	5	5.62
Language & Translation	11	2.33
Travel & Tourism	13	2.07
Coaching & Advice	14	1.93
Chatbots & Interaction	15	1.88
Communication & Dialogue	17	1.49
Science & Exploration	18	1.49
Art & Music	20	1.47
Repair & Maintenance	21	1.42
Agriculture & Farming	26	1.08
Culture & History	31	0.88
Legal & Documentation	32	0.88

Table 3.1: Prevalence of macro-topics and their associated topic IDs in the dataset

Definitions of Macro-topics:

- **Educational Content & Learning** — Topics related to educational tools, learning resources, and platforms designed to enhance knowledge and skills in various domains.

  Example: *Your personal AI tutor by Khan Academy! I'm Khanmigo Lite - here to help you with math, science, and humanities questions. I won't do your work for you, but I will help you learn how to solve them on your own. Can you tell me the problem or exercise you'd like to solve?*

- **Image & Visual Generation** — Topics that focus on tools and applications for generating, editing, and enhancing images and visuals, often using AI or machine learning techniques.

  Example: *A GPT specialized in generating and refining images with a mix of professional and friendly tone.image generator*

- **Content Creation & Writing** — Topics that encompass tools and platforms for writing, content generation, and creative expression, including blogs, articles, and storytelling.

  Example: *A creative AI assistant for content creation and writing.*

- **Business & Strategy** — Topics that include applications and tools for business management, strategic planning, and professional development.

  Example: *Get tailored strategies for your business growth and online success! Tackle everything from business plans to digital marketing trends!*

- **Technology & Programming** — Topics that are centered on programming, software development, and technological advancements, including coding practices, software tools, and innovative tech solutions.

  Example: *Delivers complete solutions to any programming problem, no matter the language and the complexity.*

- **Health & Wellness** — Topics that relate to health, fitness, wellness, and healthcare tools designed to promote physical and mental well-being.

  Example: *Create your own healthy recipes with nutritional insights, it can help you create a balance diet plan.*

- **Language & Translation** — Topics that focus on language processing, translation services, and multilingual communication tools that facilitate cross-language interaction.

  Example: *Accurate and natural like human translations in all major languages*

- **Travel & Tourism** — Topics that cover tools and resources related to travel planning, tourism, and exploration, including travel guides, booking systems, and travel recommendations.

  Example: *Expert on global travel destinations, trip planning, budget building, and exploring the world! Press T for Travel Menu.*

- **Coaching & Advice** — Topics that include personal development, coaching, and advisory services, offering guidance on various aspects of life and career.

Example: *Smart Career Coach for career advice, career tips, career coaching.*

- **Chatbots & Interaction** — Topics that involve conversational agents, chatbots, and interactive dialogue systems designed to enhance user engagement and automate communication.

  Example: *A confident and witty chatbot for fun, respectful romance chats.*

- **Communication & Dialogue** — Topics that encompass tools for enhancing communication, dialogue, and social interaction, facilitating effective conversation and information exchange.

  Example: *Creates one-question, one-answer animal dialogue comics.*

- **Science & Exploration** — Topics that are Related to scientific inquiry, research, and exploration across various fields of science, promoting discovery and knowledge expansion.

  Example: *Expert in science and science education, guiding with knowledge and pedagogy.*

- **Art & Music** — Topics that focus on artistic expression, music generation, and creative arts, including tools for creating, editing, and experiencing art and music.

  Example: *The best AI artwork generator for effortlessly creating highly detailed illustrations or photos without requiring any design expertise. This designer GPT will produce stunning images with just a simple prompt.*

- **Repair & Maintenance** — Topics that cover tools and guides for repairing, maintaining, and troubleshooting various equipment or systems, ensuring optimal functionality.

  Example: *The GPT trains building owners about the operation and maintenance of Powergate VFDs.*

- **Agriculture & Farming** — Topics that involve topics related to agriculture, farming practices, and agri-tech innovations, focusing on sustainable and efficient food production.

  Example: *A knowledgeable agriculture mentor, providing practical farming advice.*

- **Culture & History** — Topics that encompass cultural knowledge, historical information, and heritage preservation, promoting awareness and appreciation of diverse cultures and histories.

  Example: *Build history from a divergence point. As you build history, ask it to tell you stories.*

- **Legal & Documentation** — Topics that pertain to legal assistance, documentation, compliance, and regulatory information, offering guidance on legal matters and document management.

  Example: *Expert legal assistant for drafting motions, proposed orders, and other legal documents.*

Fig. 3.1 shows the differences in the distribution of macro-topics across the various platforms. We can see that "Educational Content & Learning" and "Content Creation & Writing" consistently rank among the top categories across all surveyed platforms. This underscores a strong interest by Promptware developers in tools for education and content generation within the Large Language Model (LLM) ecosystem.

GPTstore appears to focus significantly on professional and business needs, with "Business & Strategy" being its most prevalent category. This is closely followed by topics related to content creation and education, indicating a user base that leverages LLMs for business development and the enhancement of professional skills.

HuggingFace, known for its open-source contributions, emphasizes education. The data reflects a community that values learning and creativity, with a significant engagement in entertainment and technology programming, highlighting its diverse user interests.

Ora stands out with nearly half of its offerings centered around educational tools, suggesting it serves as a hub for academic and educational applications. This focus points to its use in formal learning environments or self-learning scenarios, catering to a wide range of learners.

POE distinguishes itself with a focus on visual generation and gaming, indicating a community centered around graphic design, gaming, and interactive entertainment. The creative domain seems to be a major attraction for users on this platform.

Conversely, categories such as "Chatbots & Interaction", "Culture & History", and "Legal & Documentation" appear less common than others across all platforms. While these topics are currently underrepresented, their adoption may increase as the broader LLM ecosystem continues to evolve. For example, "Chatbots & Interaction" could gain prominence with

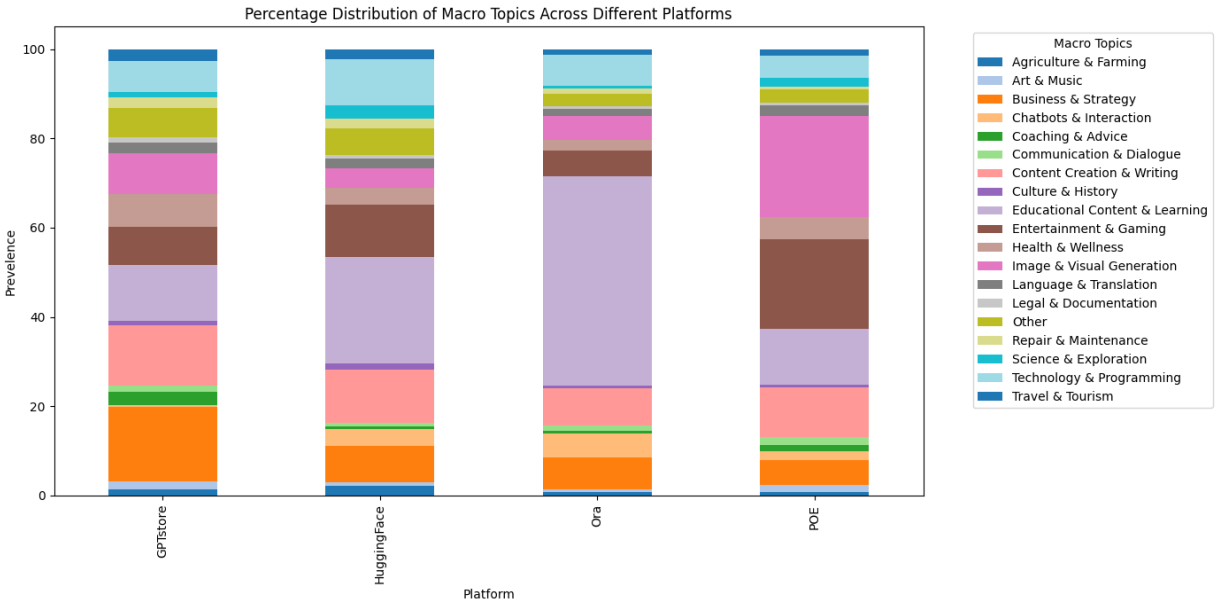


Figure 3.1: Distribution of Macro-topics across different platforms

ongoing advances in conversational AI and personalization, and "Legal & Documentation" may see growth as industries increasingly explore AI for compliance, contract review, and legal support. We hypothesize that categories that are less prominent today may become more central as technical capabilities and user needs shift over time.

Additionally, the variance in category prevalence highlights each platform's unique market positioning and strategic priorities. GPTstore's business-centric approach contrasts with HuggingFace's focus on education and open-source contributions, Ora's role as an educational resource hub, and POE's creative and entertainment orientation. These differences reflect the platforms' attempts to cater to distinct user segments and the diverse ways in which LLMs are being leveraged to meet specific needs and interests.

Overall, the findings illustrate a dynamic and varied LLM ecosystem, with each platform serving different aspects of user demand, ranging from professional development and strategic business applications to educational tools, creative arts, and entertainment. This diversity underscores the versatility of LLMs in addressing a wide array of use cases and points to evolving trends that may shape the future of FMware development and adoption.

3.2 RQ2 Results

In RQ2, we first applied a combined approach of label-based and AI-assisted classification to categorize issues from the FMware repositories. This process produced a distribution of 146,570 issues across 15 distinct categories from Github, as presented in Table 3.2. The classification reveals several trends in the types of challenges and development focuses within the FMware ecosystem.

Classification	**Count**	**Percentage (%)**
Bug Reports	42,661	30.69
Core Functionality	30,364	21.85
Miscellaneous	11,632	8.37
Documentation	10,949	7.88
Support & Questions	8,219	5.91
Feature Requests	6,706	4.82
Infrastructure & Deployment	6,584	4.74
Project Management	4,410	3.17
User Experience (UX)	3,794	2.73
Performance & Optimization	3,774	2.72
API & Integration	3,395	2.44
Quality Assurance & Testing	2,850	2.05
Platform Compatibility	1,861	1.34
Internationalization & Localization	902	0.65
Security	887	0.64

Table 3.2: Issue classification distribution

The most frequently occurring categories are "Bug Reports" (30.69%) and "Core Functionality" (21.85%), which together account for over half of the collected issues. These categories primarily reflect ongoing efforts to address technical challenges related to FMware reliability and system behavior. While this distribution may seem unsurprising — bug fixing and feature development are typically among the most common issue types in many software systems (e.g., [11]) — their dominance reinforces that FMware development shares many baseline maintenance concerns with traditional software.

What distinguishes FMware, however, becomes clearer upon deeper analysis. In the next phase, we examine these two categories using topic modeling to uncover the specific types of challenges developers face. This reveals a more nuanced picture of FMware-specific

concerns such as memory allocation failures, model loading errors, and collaborative review bottlenecks.

Other notable categories include "Support & Questions" (5.91%), reflecting a substantial need for community-driven troubleshooting and knowledge sharing, and "Feature Requests" (4.82%), which suggest continued interest in expanding FMware capabilities. These categories highlight the collaborative and evolving nature of FMware development, where users actively seek help and propose enhancements to adapt the tools to their needs.

Overall, the distribution of issue classifications provides a nuanced view of the challenges that developers encounter in FMware development. "Bug Resolution", "Core Functionality", and "Documentation" stand out as primary areas requiring ongoing attention, while categories like "User Experience" and "Infrastructure" suggest a developing interest in FMware systems.

To validate the generalization of our methodology, we further analyzed LLM-related issues on GitHub to construct the taxonomy. Unlike prior work such as that of Chen et al. [3], which directly sampled and categorized issues from official GitHub repositories of major LLM vendors — specifically, Meta AI, Google, and OpenAI — our study first applied a combined approach of label-based and AI-assisted classification to categorize a broader range of FMware-related issues. This process produced a distribution of 146,570 issues across 15 distinct categories, as shown in Table 3.2. From this classification, we identified "Bug Reports" (30.69%) and "Core Functionality" (21.85%) as the two most prevalent categories, together accounting for over 50% of all issues. To gain deeper insights, we selected these top two categories and applied topic modeling, resulting in 26,650 posts for "Bug Reports" and 24,484 posts for "Core Functionality". Unlike Chen et al. [3], who constructed a taxonomy directly from sampled issues, our approach ensures a structured categorization before topic modeling, providing a more systematic and scalable analysis of the FMware ecosystem. The taxonomy developed in our study captures a diverse range of software engineering challenges specific to FMware, complementing prior research while extending the understanding of LLM-related development issues.

3.2.1 Bug Reports Topic Modeling Results

In this section, we present the results from topic modeling on the "Bug Reports" dataset. We identified 30 distinct topics, each highlighting a specific type of issue frequently encountered by developers.

To systematically analyze the recurring issues in FMware development, we categorized the identified topics from the bug reports dataset into a structured taxonomy as **Figure 3.2**.

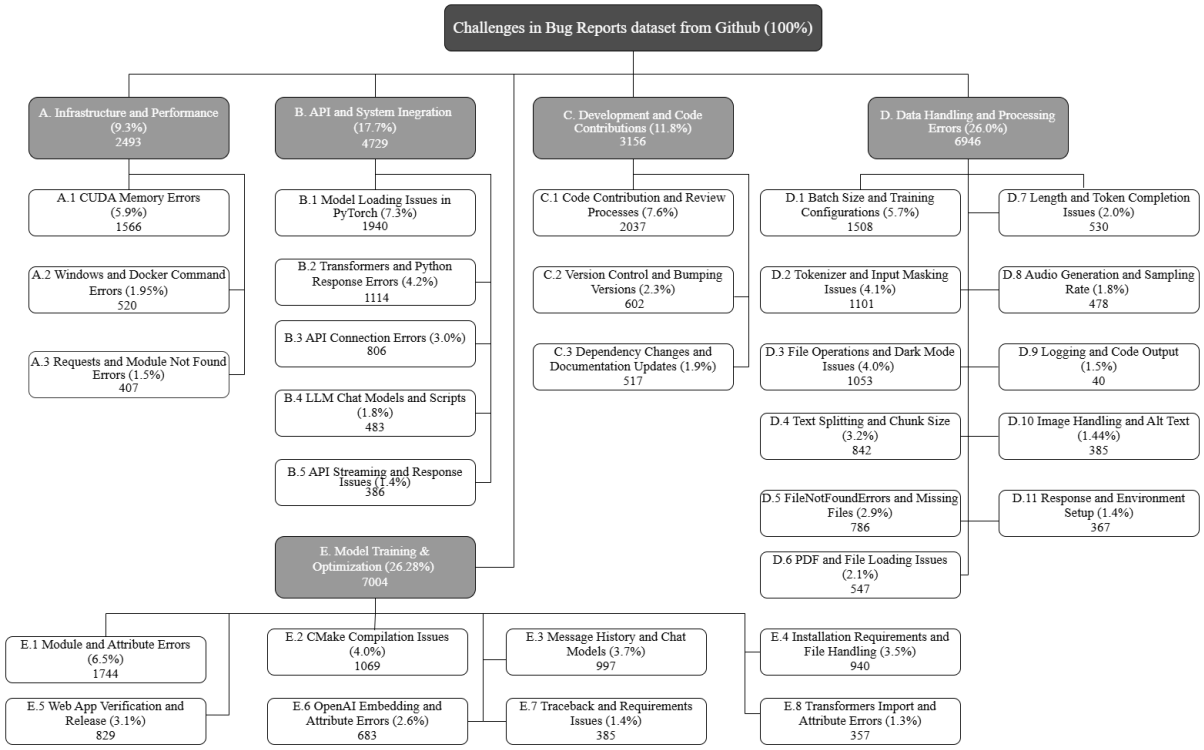


Figure 3.2: Challenge Taxonomy for Bug Reports Dataset Build from GitHub Issues

This taxonomy classifies bug-related challenges into five major categories: “Infrastructure & Hardware Issues”, “API & Model Execution”, “Development & Code Contribution Errors”, “Data Handling & Processing Errors”, and “Module & Installation Issues”. Each category encompasses distinct technical issues that developers frequently encounter, such as “CUDA Memory Errors and Hardware Compatibility Issues to API Connection Failures”, “Tokenization Problems”, “Dependency Conflicts”, and “Module Import Errors”.

3.2.2 Distribution of Challenge Topics in Bug Reports Dataset

In this section, we present the distribution of challenge topics identified in the bug reports dataset. Our analysis covered a wide range of issues, with a total of 30 distinct topics. Figure 3.2 provides an overview of the distribution of these topics.

To highlight key findings, we group the most prevalent topics under their corresponding

high-level categories and summarize their significance:

- **General Errors** are dominated by **CUDA Memory Errors (A.1, 5.9%)**, indicating persistent struggles with GPU memory constraints. These errors occur frequently in large-scale deep learning workflows, where memory optimization is crucial for model training and inference. Developers often encounter "out of memory" failures, limiting their ability to efficiently run high-performance models.
- **API and Model Execution** is primarily impacted by **Model Loading Issues in PyTorch (B.1, 7.3%)**, reflecting common challenges in restoring models from checkpoints and resolving format incompatibilities. The frequency of these errors suggests that ensuring smooth model deployment and loading remains a critical concern, particularly when integrating large FMware-based models into production environments.
- **Development and Code Contributions** see the most reports in **Code Contribution and Review Processes (C.1, 7.6%)**, emphasizing the importance of collaborative software engineering. Given that many FMware repositories rely on contributions from a broad developer community, maintaining high-quality code through structured reviews, pull request management, and version control is essential.
- **Data Handling and Processing** is largely shaped by **Batch Size and Training Configurations (D.1, 5.7%)**, highlighting difficulties in tuning hyperparameters for model training. Developers frequently face challenges in optimizing batch sizes to balance computational efficiency and GPU memory usage, underscoring the technical complexity of training deep learning models in constrained environments.
- **Module and Installation Issues** are most represented by **Module and Attribute Errors (E.1, 6.5%)**, which stem from dependency conflicts, incorrect environment configurations, and missing module imports. These errors often disrupt development workflows, requiring significant debugging efforts to resolve compatibility issues between software components.

The prominence of these topics highlights critical areas of focus within FMware development, where improvements in memory management, collaborative practices, and training setup could enhance efficiency and scalability. Other notable topics include "Model Loading Issues in PyTorch", "Module and Attribute Errors", and "Transformer Response Errors", each representing key technical challenges encountered by developers.

This distribution provides insights into the key technical and operational challenges encountered by developers in the FMware ecosystem. By focusing on the most prevalent issues, such as "Memory Management", "Code Contribution Processes", and "Training Configuration", this study offers a foundation for targeted improvements in FMware development tools and workflows.

3.2.3 Core Functionality Topic Modeling Results

In this section, we present the results from performing topic modeling on the Core Functionality dataset. We identified 30 distinct topics, each highlighting a specific type of issue frequently encountered by developers when working on core functionalities of FMware systems. To better understand the challenges faced by developers when working with the core functionalities of FMware systems, we categorized the identified topics into a structured taxonomy, presented in **Figure 3.3**. This taxonomy follows the same high-level categories as those used in the bug reports dataset (**Section 3.2.1**), ensuring consistency and enabling direct comparisons between different types of challenges encountered in FMware development. By maintaining a uniform taxonomy, we can analyze whether certain challenges are more prevalent in bug reports (indicating frequent failures and debugging needs) or in core functionality discussions (reflecting broader architectural and optimization concerns).

To highlight key findings, we group the most prevalent topics under their corresponding high-level categories and summarize their significance below:

- **Model Training & Optimization** is dominated by **Language Model Training with Pre-trained Models (A.1, 10.8%)**, indicating a strong focus on model fine-tuning, checkpoint management, and optimization techniques. Developers frequently encounter challenges in customizing pre-trained models for specific applications, ensuring training efficiency, and balancing resource constraints.
- **API & System Integration** is primarily impacted by **CPU, CUDA, and Memory Management (B.1, 6.1%)**, reflecting concerns around efficient execution of models within constrained hardware environments. This topic includes managing memory allocation, handling CUDA-related errors, and optimizing system performance when running FMware models at scale.
- **Development & Code Contribution** sees the most reports in **Documentation Updates and Testing (C.1, 6.6%)**, highlighting the need for accurate documentation and rigorous testing in open-source FMware projects. Developers often focus

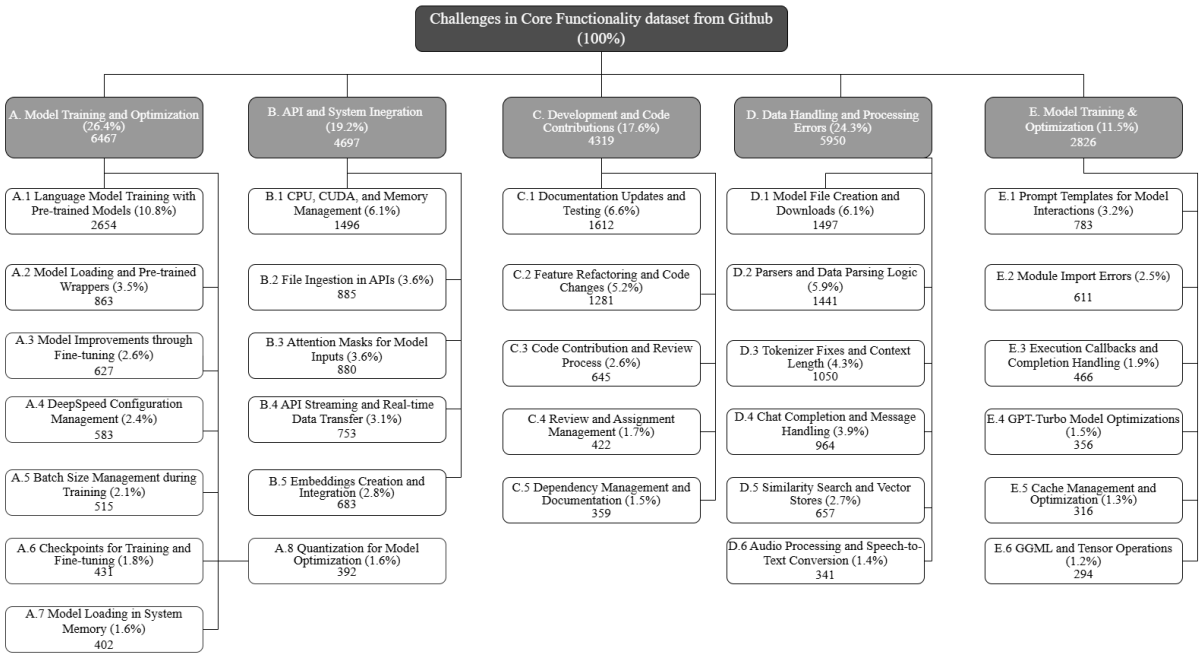


Figure 3.3: Challenge Taxonomy for Core Functionalities Dataset Build from GitHub Issues

on refining documentation to keep up with evolving features and ensuring that new contributions undergo proper validation.

- **Data Processing & Handling** is largely shaped by **Model File Creation and Downloads (D.1, 6.1%)**, emphasizing the complexities of managing pre-trained models, ensuring compatibility, and handling large model files efficiently. Given the rapid expansion of FMware-based applications, smooth handling of model assets is crucial for reducing overhead and streamlining workflows.
- **Infrastructure & Performance** is most represented by **Prompt Templates for Model Interactions (E.1, 3.2%)**, which underscores the growing importance of structured prompts in enhancing model usability and response consistency. Optimizing prompt templates ensures that language models generate reliable, meaningful, and context-aware responses across different applications.

The prominence of these topics highlights key areas of focus within FMware development, where improvements in "Model Training", "System Performance", "Documentation Practices", and "Data Handling Workflows" can enhance efficiency and scalability. Other notable topics include "Feature Refactoring and Code Changes", "API Streaming and Real-Time Data Transfer", and "Execution Callbacks and Completion Handling", each addressing essential aspects of FMware functionality.

This structured distribution provides insights into the core technical and operational challenges encountered by FMware developers. The structured breakdown provides a comprehensive view of the prevalent challenges developers face in FMware core functionality.

3.3 RQ3 Results

In RQ3, we analyze the time required to resolve various issues across different datasets to understand which topics demand more developer effort. Our study includes two primary datasets from GitHub repositories, as used in RQ2: the **Bug Reports** dataset, consisting of 26,650 issues, and the **Core Functionality Issues** dataset, consisting of 30,355 issues. For each issue, we calculate the solving time by taking the difference between the opening and closing timestamps, measured in hours. We then compute the median solving time per topic to reduce the influence of outliers and better reflect typical resolution durations. This approach provides a robust comparison of topic-level complexity and developer burden across both datasets.

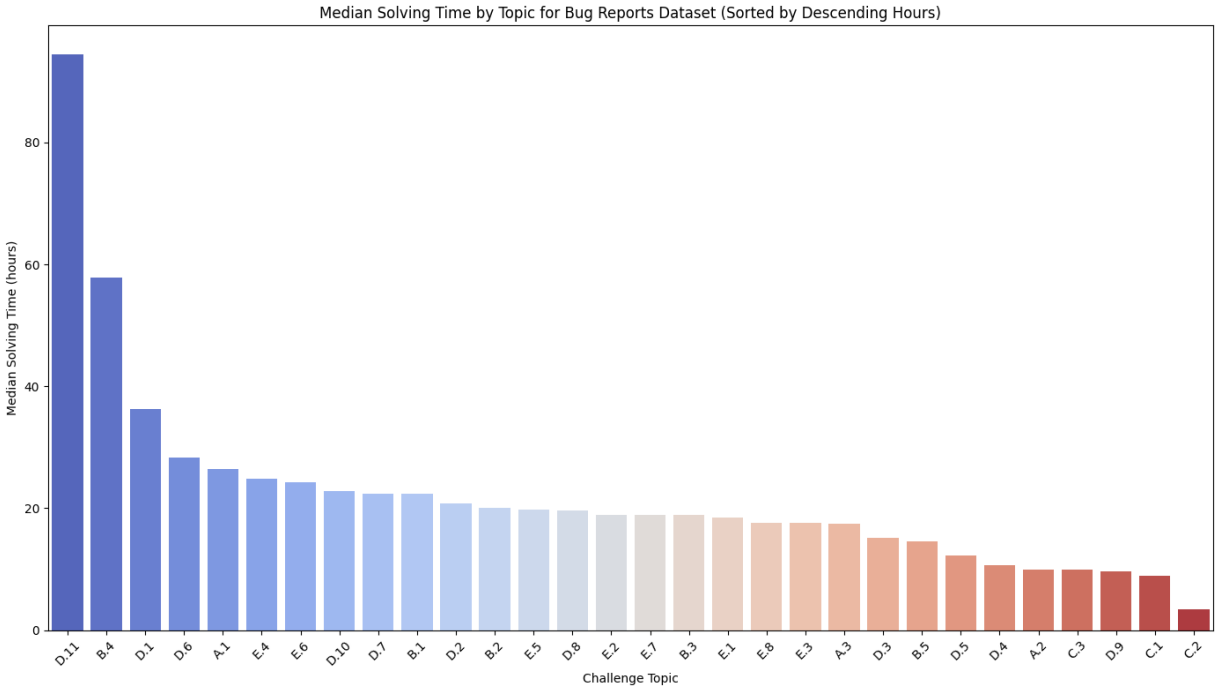


Figure 3.4: Median Solving Time by Topic for Bug Reports Dataset

3.3.1 Results for Bug Report Solving Time

Based on median solving times, three topics stand out as the most time-consuming to resolve in the Bug Reports dataset: **Response and Environment Setup (D.11, 1.4%, 367 issues)**, **LLM Chat Models and Scripts (B.4, 1.8%, 483 issues)**, and **Batch Size and Training Configurations (D.1, 5.7%, 1508 issues)**. These topics suggest complex and time-intensive development challenges in FMware.

Response and Environment Setup (D.11) has the highest median solving time. This topic includes issues related to configuring and troubleshooting environments for CUDA and PyTorch, such as version mismatches or missing components. The elevated resolution time suggests the intricacies involved in correctly setting up dependencies and runtime environments, which are often critical for FMware deployment.

LLM Chat Models and Scripts (B.4) ranks second in median resolution time. It includes tasks related to script orchestration, module integration, and managing chat-based LLM interactions. The longer solving time suggests the complexity of coordinating various components in conversational AI applications and debugging across dependent modules.

Batch Size and Training Configurations (D.1) comes third, with challenges involving tuning hyperparameters such as batch sizes for efficient model training. These tasks frequently require empirical tuning and consideration of hardware constraints, especially in large-scale training setups.

We also analyze the mean solving time for this dataset. The top three topics by mean are **LLM Chat Models and Scripts (B.4, 1.8%, 483 issues)**, **OpenAI Embedding and Attribute Errors (E.6, 2.6%, 683 issues)**, and **PDF and File Loading Issues (D.6, 2.1%, 547 issues)**. While B.4 is ranked highly in both metrics, D.11 drops significantly in the mean-based ranking—indicating the presence of extreme outliers in other categories that pull up their averages. This highlights why median-based analysis provides a more robust view of typical developer effort, especially in skewed distributions.

Together, these findings suggest that resolving infrastructure setup, conversational model integration, and training configuration issues may require extended effort. This indicates a need for better tooling, clearer documentation, and more automation for environment and training setup processes in FMware development.

3.3.2 Results for Core Function Dataset Solving Time

For the Core Functionality dataset, the top three topics with the highest median solving times are: **Language Model Training with Pre-trained Models (A.1, 10.8%,**

2654 issues), **Model Loading and Pre-trained Wrappers (A.2, 3.5%, 863 issues)**, and **Quantization for Model Optimization (A.8, 1.6%, 392 issues)**. These topics primarily fall under model training and optimization efforts.

Language Model Training with Pre-trained Models (A.1) has the highest median resolution time. This reflects the intensive and often iterative process of training and fine-tuning models. Issues under this topic include adjusting training loops, managing datasets, and ensuring convergence, which are all critical for model performance.

Model Loading and Pre-trained Wrappers (A.2) follows closely. This topic captures issues related to restoring pre-trained checkpoints and integrating wrapper interfaces. Long solving times here suggest that compatibility and version mismatches in model loading remain significant pain points in FMware development.

Quantization for Model Optimization (A.8) ranks third in time intensity. Developers face challenges optimizing models for efficient deployment—often through quantization—while ensuring that accuracy is not degraded. This process requires both domain-specific expertise and extensive experimentation.

We also analyze the mean solving time for the Core Function dataset. The top three topics by mean are **Code Contribution and Review Processes (C.3, 2.6%, 645 issues)**, **Similarity Search and Vector Stores (D.5, 2.7%, 657 issues)**, and **Prompt Templates for Model Interactions (E.1, 3.2%, 783 issues)**. This shift from A-group topics to C, D, and E groups in mean-based rankings suggests that while training-related issues are consistently time-consuming on average, developer collaboration, vector index tuning, and prompt engineering occasionally require disproportionately long resolution times — likely due to coordination delays or complex iterative debugging.

These results show that model-related engineering tasks dominate the most time-consuming FMware issues in core functionality. The findings suggest a strong need for improved tools, automation frameworks, and guidance for training, loading, and optimizing large foundation models.

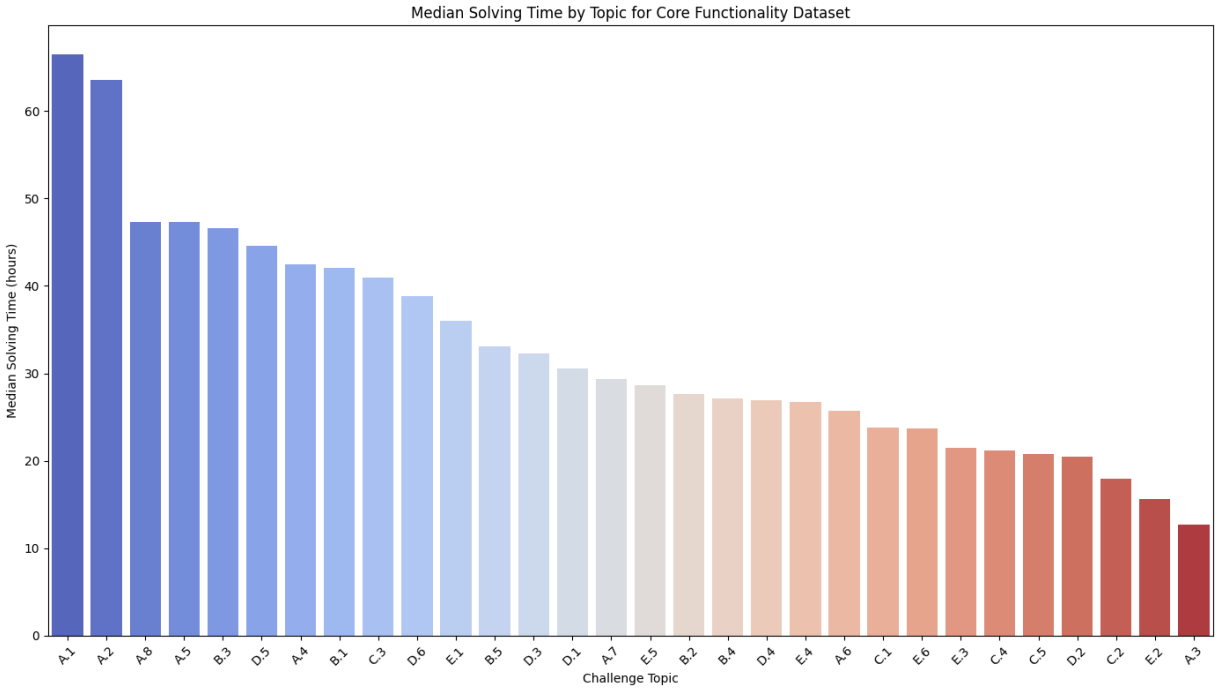


Figure 3.5: Median Solving Time by Topic for Core Functionality Dataset

Chapter 4

Discussion and Implications

Our research offers insights for a wide range of FMware practitioners, including researchers, platform developers, and tool developers. By exploring the prevalent macro-topics across different FMware platforms, we highlight key areas of focus within the ecosystem and identify potential gaps in FMware development.

4.1 Implications for Researchers

Cross-Platform Research Opportunities: One of the most critical insights from our study is the variation in macro-topic distribution across different FMware platforms, such as HuggingFace, GPTstore, Ora, and POE. This indicates that each platform serves distinct user bases and needs, ranging from business strategy to creative arts. However, the cross-platform implications of these differences remain underexplored. Further research could focus on investigating how developers and users interact across multiple FMware platforms and whether their experiences differ based on the platform they use.

Moreover, cross-platform tool interoperability remains a largely unexplored area. As developers and enterprises increasingly adopt multiple FMware solutions, understanding how tools and models can be seamlessly integrated across platforms will be key. For example, research into creating frameworks that allow for the portability of FMware models and applications from cloud-based solutions (e.g., HuggingFace) to on-premise environments (e.g., GitHub-hosted FMware) could reduce the friction developers face when migrating or deploying across different infrastructures.

Given the prevalence of topics related to Educational Content & Learning and Content Creation & Writing, there is also a need for researchers to focus on improving the usability

and collaboration within FMware platforms. A key challenge across platforms is ensuring that FMware tools are intuitive and accessible for users at different skill levels. Researchers could explore user experience (UX) design for FMware interfaces, particularly how to make complex FMware functionalities more accessible to non-experts, such as educators or content creators.

Additionally, collaborative FMware development is an area ripe for exploration. As FMware applications become more complex and involve larger teams, understanding how developers collaborate on FMware projects, share models, and manage dependencies across platforms could lead to new tools and practices that improve team-based development in the FMware ecosystem.

Benchmarking and Evaluation Frameworks for FMware: Finally, our study identifies a gap in the benchmarking and evaluation of FMware tools. While existing FMware platforms cater to diverse use cases, there is no standardized framework for evaluating the effectiveness or performance of FMware models across different platforms. Future research could work on developing evaluation benchmarks that assess FMware performance across multiple dimensions, such as efficiency, scalability, and user satisfaction. These benchmarks would help researchers and practitioners alike to better understand which FMware platforms and tools are most effective for particular tasks and could guide future development efforts.

Chapter 5

Threats to Validity

In this section, we discuss potential threats to the validity of our study and how they could impact our findings.

Internal Validity: Several factors may introduce biases in our topic modeling and classification approach. First, **topic modeling limitations** may arise due to the use of BERTopic, which is sensitive to pre-processing choices, hyperparameter selection, and the underlying embedding model. As a result, the extracted topics might not always reflect the true concerns of FMware developers. Second, **the accuracy of the data labeling** could be affected by AI-assisted classification of issues, particularly for unlabeled GitHub issues, leading to possible misclassification errors that affect the accuracy of the topic distribution. Third, **preprocessing loss of information** may have occurred, as the removal of certain stop words such as "error" or "bug" could have unintentionally stripped meaningful context from the discussion of the issue.

External Validity: The extent to which our findings generalize beyond the analyzed dataset is another important consideration. **Generalizability across the FMware ecosystem** is a potential concern, as our data set consists primarily of GitHub-hosted FMware projects, which may not fully represent the challenges faced in cloud-based platforms such as HuggingFace, Poe, or Ora. Differences in infrastructure, deployment environments, and user interactions may lead to distinct issue distributions across platforms. Additionally, **community-specific biases** exist, as our data set primarily captures developer-reported concerns on GitHub, potentially underrepresenting user-centric issues (e.g., usability and accessibility challenges) that are discussed more frequently on forums like OpenAI Discussions.

Construct Validity: The consistency of our taxonomy across different datasets presents

a potential threat to construct validity. While we maintain the same upper categories for both the Bug Reports dataset and the Core Functionality dataset, ensuring comparability, **taxonomy consistency across datasets** may overlook domain-specific differences in issue types. Certain challenges that are unique to core functionality discussions may not align perfectly with those found in bug reports, potentially leading to misclassification or loss of granularity.

Reliability: The reproducibility of our findings is subject to several factors. **Reproducibility of results** is influenced by BERTopic hyperparameter settings, AI-assisted classification methods, and dataset selection. Different runs with varying parameters or data sources may yield slight variations in topic distributions. Additionally, **changes in the FMware landscape over time** present a challenge, as the dataset reflects a snapshot of current FMware development practices. As FMware tools and frameworks evolve, new challenges may emerge that are not captured in this study, requiring ongoing investigation to track shifting trends in FMware development.

Chapter 6

Summary

This study presents the first comprehensive empirical analysis of FMware development —software systems built around foundation models —by examining both cloud-based Promptware platforms and on-premise GitHub-hosted FMware repositories. Our investigation addressed three core research questions: identifying prevalent topics within FMware platforms, uncovering key development challenges, and determining which types of issues require the most time to resolve.

While we use resolution time as a proxy to understand developer workload patterns, we acknowledge that time does not directly measure developer effort. Future work may explore complementary measures such as code change size or discussion volume to better estimate effort.

Our results show that FMware platforms focus heavily on educational tools, content creation, and business applications, reflecting the broad applicability of foundation models across diverse domains. From a developer perspective, we identified critical technical challenges, including memory constraints, dependency management, tokenizer configuration, and collaborative development workflows. Notably, issues related to code contribution, similarity search, and prompt engineering are the most time-consuming to resolve, underscoring the complexity of building robust and scalable FMware systems.

By analyzing over 25,000 FMware applications and more than 140,000 GitHub issues, this study provides insights for researchers, practitioners, and tool developers. It highlights the need for improved tooling, better debugging support, and more effective workflows to support the growing FMware ecosystem. As FMware continues to evolve, addressing these challenges will be critical to unlocking the full potential of foundation models in software engineering.

References

[1] Toufique Ahmed, Christian Bird, Premkumar Devanbu, and Saikat Chakraborty. Studying LLM performance on closed-and open-source data. *arXiv preprint arXiv:2402.15100*, 2024.

[2] John L Campbell, Charles Quincy, Jordan Osserman, and Ove K Pedersen. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. *Sociological methods & research*, 42(3):294–320, 2013.

[3] Xiang Chen, Chaoyang Gao, Chunyang Chen, Guangbei Zhang, and Yong Liu. An empirical study on challenges for LLM application developers. *ACM Transactions on Software Engineering and Methodology*, January 2025.

[4] Yihao Chen, Eduardo Fernandes, Bram Adams, and Ahmed E Hassan. On practitioners' concerns when adopting service mesh frameworks. *Empirical Software Engineering*, 2023.

[5] Filipe Roseiro Cogo, Gopi Krishnan Rajbahadur, Dayi Lin, and Ahmed E. Hassan. A tutorial on software engineering for FMware. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, FSE 2024, page 710–712, New York, NY, USA, 2024. ACM.

[6] Github. *Installing GitHub Enterprise Server on VMware*, 2023. Accessed: 2024-10-02.

[7] Github. *Installing GitHub Enterprise Server on VMware*, 2024. Accessed: 2024-10-02.

[8] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv preprint arXiv:2203.05794*, 2022.

[9] Ahmed E. Hassan, Dayi Lin, Gopi Krishnan Rajbahadur, Keheliya Gallaba, Filipe Roseiro Cogo, Boyuan Chen, Haoxiang Zhang, Kishanthan Thangarajah, Gustavo Oliva,

Jiahuei (Justina) Lin, Wali Mohammad Abdullah, and Zhen Ming (Jack) Jiang. Rethinking software engineering in the era of foundation models: A curated catalogue of challenges in the development of trustworthy FMware. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, FSE 2024, page 294–305, New York, NY, USA, 2024. ACM.

[10] Ahmed E. Hassan, Gustavo A. Oliva, Dayi Lin, Boyuan Chen, Zhen Ming, and Jiang. Towards AI-native software engineering (SE 3.0): A vision and a challenge roadmap, 2024.

[11] Steffen Herbold, Alexander Trautsch, Fabian Trautsch, and Benjamin Ledel. Problems with szz and features: An empirical study of the state of practice of defect prediction data collection. *Empirical Software Engineering*, 27(2):42, 2022.

[12] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Transactions on Software Engineering and Methodology*, 55(12), March 2023.

[13] Swaranjali Jugran, Ashish Kumar, Bhupendra Singh Tyagi, and Vivek Anand. Extractive automatic text summarization using spacy in python & nlp. In *2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, pages 582–585, 2021.

[14] Microsoft Learn. *Configure GitHub Enterprise Server on Azure VMware Solution*, 2024. Accessed: 2024-10-02.

[15] Dayi Lin, Filipe Roseiro Cogo, Gopi Krishnan Rajbahadur, and Ahmed E. Hassan. Technical brief on software engineering for FMware. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, ICSE-Companion '24, page 431–433, New York, NY, USA, 2024. ACM.

[16] Dayi Lin, Filipe Roseiro Cogo, Gopi Krishnan Rajbahadur, and Ahmed E. Hassan. Technical brief on software engineering for FMware. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, ICSE-Companion '24, page 431–433, New York, NY, USA, 2024. ACM.

[17] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36, 2024.

[18] Dongxun Su, Yanjie Zhao, Xinyi Hou, Shenao Wang, and Haoyu Wang. Gpt store mining and analysis. *arXiv preprint arXiv:2405.10210*, 2024.

[19] Shaheen Syed and Marco Spruit. Full-text or abstract? examining topic coherence scores using latent dirichlet allocation. In *2017 IEEE International conference on data science and advanced analytics (DSAA)*, pages 165–174. IEEE, 2017.

[20] Vmware. Deploying an LLM powered coding assistant on VMware private AI. 2023. Accessed: 2024-10-02.

[21] Zitao Wang. Fmware replication package. `https://github.com/Nick77666/FMware-replication-package`, 2025. Accessed: April 2025.

APPENDICES

Appendix: Stop Word Lists Used in Topic Modeling

Stop Words Used in RQ1: PromptWare Descriptions

For RQ1, we removed general English nouns and vague domain terms that did not help differentiate topics, alongside frequent but uninformative platform-specific words. Below is the final stop word list used after manual review and coherence tuning.

General Content Words Removed:

ability, abilities, accident, accidents, acknowledgement, action, actions, activities, activity, advantage, advantages, alternative, alternatives, announcement, announcements, anomaly, anomalies, answer, answers, appreciation, approach, approaches, article, articles, assistance, author, behavior, behaviour, benefit, benefits, bit, bits, body, case, cases, cause, causes, challenge, challenges, change, changes, check, choice, choices, collection, com, combination, concern, concerns, confirmation, confusion, consideration, context, couple, couples, course, courses, cross, day, days, demand, desire, detail, details, difference, differences, difficulties, difficulty, disadvantage, disadvantages, discrepancies, discrepancy, discussion, discussions, dislike, distinction, effect, end, enquiries, enquiry, evidence, example, examples, exception, exceptions, existence, exit, expectation, experience, expert, experts, explanation, explanations, fact, facts, favorite, favorites, feedback, feedbacks, forum, forums, future, goal, goals, guarantee, guidance, guideline, guide, guides, guy, guys, harm, hello, help, hour, hours, idea, ideas, individual, individuals, info, information, inquiries, inquiry, insight, intelligence, intent, interest, introduction, investigation, invitation, issues, kind, kinds, lack, learning, level, levels, look, looks, lot, lots, luck, major, manner, manners, manual, mark, means, meaning, method, methods, minute, minutes,

month, months, need, needs, number, numbers, offer, one, ones, opinion, opinions, outcome, part, parts, past, people, person, persons, perspective, perspectives, place, places, point, points, post, posts, practice, practices, problem, problems, product, products, proposal, proposals, purpose, purposes, py, qa, questions, reason, reasons, result, results, scenario, scenarios, science, second, seconds, section, sense, shortcoming, shortcomings, show, shows, situation, software, solution, solutions, start, study, stuff, success, summary, summaries, surprise, support, supports, task, tasks, technique, techniques, technologies, technology, term, terms, tip, tips, thank, thanks, thing, things, thought, thoughts, three, title, today, tomorrow, total, trouble, troubles, truth, try, tutorial, tutorials, two, understand, understanding, usage, use, user, users, uses, view, viewpoint, way, ways, weakness, weaknesses, week, weeks, word, words, work, workaround, workarounds, works, yeah, year, years, yesterday

Platform-Specific and Domain-Redundant Words Removed:

GPT, gpt, gpts, store, gptstore, chatgpt, bot, bots, com, p, post, a, openai, open, ai, AI, https, http, assistant, assistants, knowledge, custom, think, star, want, try, trying, year, kid, chat, chats

These words were identified through term frequency analysis and manually reviewed to exclude overly generic or FMware-branded terms that reduced topic distinctiveness.

Stop Words Used in RQ2: GitHub Issues

For RQ2, we removed both generic English terms and GitHub- or FMware-specific words that frequently appeared in issue discussions but did not meaningfully contribute to topic separation. This included discussion-related filler words, platform references, and terms overly tied to specific bug descriptions (e.g., “bug”, “fix”, “error”).

Platform-Specific and Redundant Discussion Terms:

GPT, gpt, gpts, store, gptstore, chatgpt, bot, bots, gpt4, gpt-4, com, p, post, a, openai, open, ai, AI, https, http, assistant, assistants, knowledge, custom, think, star, want, try, trying, year, kid, chat, chats, issue, issue number, issues, issues_number, issue_number, image, images, video, creator, guidance, friend, wingman, photo, invest, investment, game, play, fitness, therapy, seo, learn, writing, essay, search, url, urls, index, error, fix, fixe, fixed, bug, bugs, buggy

General Content Words Removed:

> ability, accident, acknowledgement, action, activities, advantage, article, assistance, author, behavior, benefit, bit, cause, challenge, check, choice, com, detail, discrepancy, discussion, dislike, effect, example, exception, experience, expert, fact, feedback, goal, guideline, idea, info, inquiry, method, minute, need, number, offer, part, point, problem, reason, result, second, solution, support, task, technique, term, thing, tutorial, usage, way, week, work

These words were selected through manual review and topic coherence evaluation. Diagnostic keywords such as "error" and "bug" were removed only after verifying their overrepresentation in nearly all issue types, limiting their value in distinguishing topics.