UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dept. of Computer Systems and Computation

CreatAIlity: A collaborative storytelling framework for fostering children's narrative creativity based on Large Language Models.

Master's Thesis

Master's Degree in Software Systems Engineering and Technology

AUTHOR: Carvalho, Alexandre Dervazi

Tutor: Jaén Martínez, Francisco Javier

ACADEMIC YEAR: 2023/2024

UNIVERSIDAD POLITÉCNICA DE VALENCIA

MASTER'S THESIS

CreatAIlity: A collaborative storytelling framework for fostering children's narrative creativity based on Large Language Models

Author:
Alexandre Dervazi CARVALHO

Supervisor:
Dr. Francisco Javier JAÉN MARTÍNEZ

A thesis submitted in fulfillment of the requirements for the degree of Master's Degree in Software Systems Engineering and Technology

in the

Department of Computer Systems and Computation

September 9, 2024

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Abstract

ETSINF - Higher Technical School of Computer Science

Department of Computer Systems and Computation

Master's Degree in Software Systems Engineering and Technology

CreatAIlity: A collaborative storytelling framework for fostering children's narrative creativity based on Large Language Models

by Alexandre Dervazi CARVALHO

The development of the CreatAIlity framework represents a significant advancement in the integration of artificial intelligence and educational technology, specifically designed to foster creativity in children through guided storytelling. This thesis explores the combination of Large Language Models (LLMs) and Finite State Machines (FSMs) by leveraging important concepts of microtasks, prompt chaining and *incipits* to create a tool that balances the spontaneity of AI-driven content generation with the educational need for coherence and structure in narrative construction. The modular and flexible design of CreatAIlity allows for experimentation with other flow control techniques while also supports integration with diverse tech stacks, making it adaptable to different educational contexts and applications. The framework's design and implementation were validated through a proof-of-concept application namely StoryBuilder, evaluated using tools such as the System Usability Scale (SUS), Game Experience Questionnaire (GEQ), and NASA Task Load Index (NASA-TLX). The positive results from these evaluations affirm the framework's potential to innovate in educational practices by making learning more engaging and interactive. This work not only introduces a novel approach to AI-driven storytelling but also provides a foundation for future research in exploring new ways to harness the capabilities of AI for educational and creative purposes.

Acknowledgements

I would like to sincerely thank my family for their unwavering support and for adapting to all the changes that came with this journey. Their belief in me has been a pillar of strength throughout. I am especially grateful to my wife, Cris, and my son, Theo, whose encouragement and patience have meant so much to me. Theo's enthusiasm for my work has been a constant source of motivation, reminding me to always strive for my best. Children often impart the most profound lessons, if we take the time to listen, and he has certainly done so for me.

I would also like to express my deep gratitude to my advisor, Profesor Dr. Francisco Javier Jaén Martínez, for his constant guidance and mentorship. His valuable advice, insightful suggestions, and constructive feedback have have been instrumental in shaping this work. I feel incredibly lucky to have had the chance to learn from his expertise, which has been a crucial part of both this work and my growth.

I also extend my special gratitude to all the professors and colleagues I had the pleasure of learning from during the master's course. The opportunity to share in their knowledge not only enriched my academic experience but also gifted me the joy of discovering new things.

Lastly, I am truly thankful to André, Diogo, Fátima, Helena, Ivan, Karen, Marcos, Nilton and Thiago for their help in reviewing and evaluating this work. Their thoughtful insights and different perspectives have been vital in enriching this thesis, pushing it beyond what I could have achieved alone. The care with which they engaged with my research speaks to their generosity and the value of collaborative thinking.

All in all, I feel incredibly lucky.

Contents

Abstract iii

Acknowledgements v

1 Introduction 1

2 Literature Review 7
- 2.1 Large Language Models (LLM) . . . 7
- 2.2 Microtasks and Prompt Chaining . . . 8
- 2.3 Human-AI Collaboration in Creative Writing and *Incipits* . . . 9
- 2.4 Finite State Machines with Stack-Based Control . . . 11

3 System Architecture and Design 13
- 3.1 Overview of CreatAIlity Architecture . . . 13
- 3.2 Design of the CreatAIlity Framework for Storytelling . . . 14
  - 3.2.1 Integration with LLMs and Prompt Chaining . . . 14
    - Microtasks and Prompt Chaining . . . 14
    - Utilizing *Incipits* . . . 16
    - Prompt Chaining . . . 16
    - Modular and Flexible Design . . . 16
  - 3.2.2 Resource-based State Management . . . 16
    - Resource Definition . . . 16
    - State Transitions . . . 17
  - 3.2.3 Stack-based Control Mechanism . . . 17
    - State Stack . . . 17
- 3.3 Framework Modules Description and Specification . . . 18
  - 3.3.1 Core Library . . . 19
    - Input-Output Handling Module . . . 19
    - LLM Handling Module . . . 24
    - State Machine Module . . . 24
    - Management Module . . . 29
    - AppConfig Module . . . 35
    - EventBroker Module . . . 36
    - Logging Module . . . 36
  - 3.3.2 Application Development Facilities . . . 37
    - State Update Handlers Module . . . 37
    - WebSocket and Flask Bootstrapping Module . . . 38
    - Tk Bootstrapping Module . . . 38
    - Tk Debug Window Bootstrapping Module . . . 39
- 3.4 Future Work . . . 39
  - 3.4.1 Enhanced Testing Strategies . . . 39
    - Unit Testing . . . 40

Integration Testing 40
Performance Testing 40
3.4.2 Improved Error Handling and Fault Tolerance 40
Error Handling 40
Fault Tolerance 40
3.4.3 Scalability and Performance Optimization 40
Scalability Considerations 41
Performance Optimization 41
3.4.4 Security Enhancements 41
Data Security 41
Access Control 41
3.4.5 Proof of Concept Considerations 41
3.5 Conclusion 42

4 Implementation and Evaluation **43**
4.1 Framework Implementation 43
4.1.1 Technical Choices 43
4.1.2 Challenges and Limitations 44
Architecture 44
Programming Language and Development Process 45
Large Language Model 46
4.1.3 Code Metrics 48
4.2 StoryBuilder PoC Application 48
4.2.1 Application Overview 48
4.2.2 Developer Responsibilities 48
User Interface Development 49
4.2.3 State/Resource and Behavior Design 49
4.2.4 Interfaces and Interaction 52
Console 52
Desktop 52
Web 56
4.3 Evaluation 57
4.3.1 Methodology 57
4.3.2 Evaluation Tools 58
System Usability Scale (SUS) 58
Game Experience Questionnaire (GEQ) 59
NASA Task Load Index (NASA-TLX) 59
4.4 Results and Discussion 60
4.4.1 System Usability Scale (SUS) 60
SUS Score Summary 60
Interpretation of SUS Items 60
Overall Interpretation 61
Final Notes 61
4.4.2 Game Experience Questionnaire (GEQ) 62
GEQ Dimensions Score Summary 63
Interpretation of GEQ Dimensions 63
Overall Interpretation 64
Final Notes 64
4.4.3 NASA Task Load Index (NASA-TLX) 65
NASA-TLX Score Summary 66
Interpretation of NASA-TLX Dimensions 66

Overall Interpretation 67
Final Notes 67

5 Conclusion 69
5.1 Summary of Achievements 69
5.2 Contributions to the Field 69
5.3 Future Research Directions 70
5.4 Potential Impact 70
5.5 Practical Applications 71
5.6 Final Notes 71

A CreatAIlity Framework and PoC Application Code Metrics 73
A.1 Raw Metrics 73
A.2 Cyclomatic Complexity 73
A.3 Maintainability Index 75

B Evaluation Questionnaires 77
B.1 System Usability Scale (SUS) 77
B.2 Game Experience Questionnaire (GEQ) 78
B.3 NASA Task Load Index (NASA-TLX) 78

Bibliography 81

List of Figures

3.1 CreatAIlity i* model (Dependencies kept on Actors for clarity) 15
3.2 Core Modules and Abstractions 18
3.3 Event Broker Module Connections 18
3.4 Event Types, Messages and Responses 18
3.5 IO Input Module Class Diagram 20
3.6 IO Input Module Events Class Diagram 21
3.7 IO Output Module Class Diagram 22
3.8 IO Output Module Events Class Diagram 23
3.9 LLM Module Class Diagram 25
3.10 LLM Module Events Class Diagram 26
3.11 State Machine Module Class Diagram 27
3.12 State Machine Module Events Class Diagram 28
3.13 State Manager Coordinator, Subordinate Managers and Event Broker Class Diagram 30
3.14 State Manager Coordinator, Subordinate Managers and Event Broker Events Class Diagram 31

4.3 StoryBuilder Console Application 52
4.1 State diagram for Storybuilder PoC 53
4.2 Resource flow diagram for Storybuilder PoC 54
4.4 StoryBuilder Desktop (Tk) Application 55
4.5 StoryBuilder Desktop (Tk) Application Debug Window 56
4.6 StoryBuilder Web Application 57
4.7 StoryBuilder Web Application Debug Modal 57
4.8 SUS Score Distribution 62
4.9 SUS Response Proportion 62
4.10 GEQ Score Radar 65
4.11 GEQ Mean-Deviation Radar 65
4.12 NASA-TLX Score Dimensions 67

List of Tables

4.1 Summary of SUS Score and Standard Deviation 60
4.2 Associated Question Items and GEQ Dimensions Scores 63
4.3 Breakdown of Mean Scores and Standard Deviations for NASA-TLX Dimensions 66

A.1 Raw Metrics Summary 73
A.2 Cyclomatic Complexity Reference Risk Levels (Lacchia, n.d.) 73
A.3 Cyclomatic Complexity Scores for Classes 74
A.4 Maintainability Index Reference Table (Lacchia, n.d.) 75
A.5 Maintainability Index Scores for CreatAIlity and PoC Applications . . 75

B.1 System Usability Scale Evaluation Statements 77
B.2 System Usability Scale Evaluation Answers 77
B.3 Game Experience Questionnaire Evaluation Statements 78
B.4 Game Experience Questionnaire Evaluation Answers 78
B.5 NASA Task Load Index Evaluation Questions 78
B.6 NASA Task Load Index Evaluation Answers 79

To my wife, Cris, and my son, Theo. Your love, encouragement, and unwavering belief in me have been my greatest sources of strength and inspiration.
To my parents, who invested in their children and believed that we could achieve even more than they did - your values and love have shaped everything I am today.

Chapter 1

Introduction

The advent of large language models (LLMs) has revolutionized natural language processing (NLP), enabling a wide range of applications from automated content generation to advanced conversational agents (Lin, 2024). Among these, the potential for LLMs to enhance educational tools is particularly promising. The integration of LLMs and AI in education enhances the learning experience by providing personalized and immediate feedback, supporting student-teacher interactions, and facilitating communication. These tools can answer routine student queries, help in understanding course material, and even provide support during online and hybrid learning models. AI-powered educational tools can customize learning experiences based on individual student needs, preferences, and abilities, thereby enhancing student engagement, motivation, and the overall quality of learning (Kooli, 2023; Chen, Chen, and Lin, 2020).

In the context of education and child development, storytelling is a critical component of early childhood education as it fosters creativity, language development, and cognitive skills (Nack and Gordon, 2016). While traditional storytelling methods are effective, they could also be significantly enhanced through interactive and adaptive technologies (Lin, 2024). According to Sawyer, 2003 and Addone, Palmieri, and Pellegrino, 2022, engaging in storytelling allows children to experiment with language, expand their vocabulary, and develop narrative skills that are essential for literacy. Additionally, storytelling stimulates imaginative thinking, enabling children to create and explore new worlds, which enhances their cognitive flexibility and problem-solving abilities. This process of creating and organizing stories helps children understand and internalize the structure of narratives, promoting better comprehension and communication skills.

Storytelling, particularly interactive storytelling, also plays a crucial role in the socio-emotional development of children. Engaging in narrative creation and enactment allows children to understand and express emotions, enhancing their social competence and emotional recognition. Tools like the Tales Toolkit provide a structured yet flexible framework for children to explore various scenarios and emotions in a safe environment. Studies have shown that such interactive storytelling activities significantly boost children's school readiness, social competence, and overall socio-emotional skills, which are essential for their academic success and personal development (Jones Bartoli, 2018).

Interactive storytelling supports high-quality interactions between children and their peers or adults, which is essential for developing social-emotional skills. These interactions help children learn to express themselves, listen to others, and understand different perspectives. Oral storytelling, in particular, has been shown to foster emotional literacy and empathy by allowing children to identify with characters and situations, thereby gaining deeper insights into their own and others' emotional states (Hibbin, 2016). This process helps children build a stronger sense of identity and emotional regulation, which are foundational for their future social and academic endeavors. Technology-enhanced interactive storytelling further engages children, providing immersive and interactive experiences that support their socio-emotional development by facilitating emotion regulation and empathy in a practical, engaging manner (Catala, Gijlers, and Visser, 2023).

The use of templates (*incipits*) in storytelling tools provides a methodical yet flexible framework that can guide children through the process of narrative creation. According to Addone, Palmieri, and Pellegrino, 2022, templates serve as scaffolding resources that support both educators and learners by providing a starting point for story creation. This approach not only helps children who might struggle to start a story but also encourages them to think creatively within a structured environment. The study highlighted that these *incipits* can significantly enhance children's engagement and creativity by offering initial prompts and allowing for individual or collaborative story development.

However, leveraging LLMs for structured tasks such as story generation presents unique challenges. One of the primary issues is the inherent non-determinism of LLMs, which can lead to inconsistent and unpredictable outputs (Ouyang et al., 2023; Lin, 2024). This unpredictability is problematic in any systematic scenario, especially in educational contexts where coherence and structure are essential. In educational storytelling, maintaining a coherent narrative is crucial for both engagement and cognitive development. Inconsistent outputs can disrupt the flow of the story, making it difficult for children to follow and engage with the narrative. Furthermore, unpredictability can undermine the educational goals of storytelling, such as teaching narrative structure, logical progression, and cause-and-effect relationships.

Given the challenges posed by the non-deterministic nature of LLMs in structured tasks like story generation, it becomes imperative to find methods that can harness their creative potential while ensuring consistency and coherence. This necessity is especially pronounced in educational contexts, where the primary goal is to foster an organized and engaging learning environment. To start, integrating some control technique which is able to ensure steps and how to transit between them can play an important part in ensuring consistency. Finite state machines (FSMs) with LLMs could offer a promising solution, providing a structured backbone for a framework that guides the narrative flow while harnessing the generative power of LLMs (Liu et al., 2024; Alagar and Periyasamy, 2011).

The FSM implementation discussed uses a requires/provides approach to dynamically link states based on their dependencies and outputs. This method ensures that each state is entered only when all its required resources are available, promoting a coherent and logical progression through the storytelling process. By defining clear dependencies and outputs, the FSM can handle the complexity of narrative creation in a structured manner. This dynamic linking of states makes the approach highly suitable for educational applications, as it provides a robust framework for guiding users through the creation process while maintaining consistency and coherence in the generated stories. This ensures that each part of the story builds logically on the previous parts, enhancing the overall narrative flow and making it easier for children to engage with and understand the story.

By leveraging FSMs, one can ensure that each part of the story adheres to a coherent structure while allowing flexibility within each state. This approach helps maintain logical flow and consistency, addressing the non-determinism of LLMs by constraining their generative outputs within the predefined states and transitions of the FSM. This structure supports children's narrative creativity by providing a guided yet flexible framework that adapts to their input and keeps the story development on track (Sklyarov, 1999; Alagar and Periyasamy, 2011).

On another topic, the work of (Wu, Terry, and Cai, 2022) on AI Chains provides additional insights into managing the non-determinism of LLMs through the concept of chaining LLM prompts. Building on the foundational work of Cai, Iqbal, and Teevan, 2016 in "Chain Reactions: The Impact of Order on Microtask Chains," which examined how the order of microtasks affects performance and cognitive load, they have demonstrated how breaking down complex tasks into smaller, manageable sub-tasks and chaining the outputs together can improve transparency, controllability, and overall quality of LLM outputs.

In the context of this discussion, this notion of prompt chaining aligns well with the integration of FSMs, specially when linking with *incipits*. FSMs can serve as the structural backbone, ensuring coherence and logical progression in the narrative, while the chaining of smaller, *incipits*-based prompts can handle the creative and generative aspects of storytelling. By structuring the interaction as a series of smaller, well-defined tasks, the guided story building process can guide the narrative development in a controlled manner, enhancing the overall user experience and ensuring the educational objectives are met.

Despite advancements in NLP and the capabilities of LLMs, a significant gap remains in developing tools that effectively integrate these technologies for structured educational tasks. The non-deterministic nature of LLMs often results in outputs that lack coherence and consistency, making them less suitable for applications requiring structured guidance, such as children's storytelling (Renzi et al., 2023; Tao, Wei, and Wang, 2008; Gmeiner and Yildirim, 2023). By leveraging FSMs and the chaining approach, the aim would be to overcome these challenges, providing a reliable and engaging storytelling experience.

This work explores the development and validation of "CreatAIlity," a collaborative storytelling framework designed to foster children's narrative creativity. By combining FSMs with the dynamic chaining of microtasks states defined by an *incipit* and LLMs, CreatAIlity aims to provide an engaging, interactive, and educational storytelling tool that supports children in crafting coherent and imaginative narratives (Renzi et al., 2023; Liu et al., 2024).

By combining FSMs with the principles of *incipits* and chaining LLM prompts, CreatAIlity not only aims to create a coherent and structured storytelling framework but also to enhance the transparency and controllability of AI-generated narratives, thereby significantly improving the educational value and engagement of storytelling tools for children.

It also addresses the following research problem: How can the integration of FSMs with LLMs be utilized to create a collaborative storytelling framework that supports children's narrative creativity while ensuring coherence and consistency in the generated stories?

The primary objectives of this study are:

1. **Design and Implement a Resource-based FSM Framework**: Develop a Finite State Machine (FSM) framework that uses a resource-based approach with stack-based control to manage the flow and structure of storytelling processes.

2. **Integrate Large Language Models (LLMs)**: Incorporate the use of advanced LLMs, specifically the Ollama Llama3 model, to generate creative content within the storytelling framework.

3. **Utilize Prompt Chaining and Microtasks**: Implement the technique of chaining LLM prompts and decomposing tasks into microtasks to enhance coherence, manage complexity, and improve the quality of generated narratives.

4. **Develop a Proof-of-Concept Application**: Create a functional proof-of-concept (PoC) application that demonstrates the practical implementation of the resource-based FSM and its integration with LLMs for storytelling.

5. **Evaluate System Performance and Usability**: Assess the performance, usability, and educational value of the PoC application through user testing and feedback, ensuring it meets the objectives of fostering creativity and supporting narrative development in children.

6. **Ensure Educational Alignment**: Align the storytelling framework with educational goals, ensuring that the generated narratives support language development, cognitive skills, and socio-emotional growth in children.

7. **Document and Share Findings**: Provide comprehensive documentation of the development process, system architecture, and findings, including a detailed description of the code and setup instructions, to facilitate replication and further research.

8. **Explore Modularity and Extensibility**: Demonstrate the modularity and extensibility of the framework, showing how it can be adapted or extended to accommodate new storytelling elements, educational goals, or technological advancements.

This study contributes to the fields of NLP, educational technology, and interactive storytelling in several ways:

- **Novel Resource-based FSM Framework**: Introduces a novel Resource-based Finite State Machine (FSM) framework with stack-based control that effectively manages the flow and structure of interactive storytelling processes.

- **Integration of LLMs in Structured Frameworks**: Demonstrates the integration of advanced Large Language Models (LLMs), specifically the Ollama Llama3 model, within a structured FSM framework to generate coherent and creative narratives.

- **Application of Prompt Chaining and Microtasks**: Applies the technique of chaining LLM prompts and decomposing tasks into microtasks, enhancing the coherence and manageability of complex storytelling tasks.

- **Development of a PoC Application**: Provides a proof-of-concept (PoC) application that showcases the practical implementation and benefits of the resource-based FSM approach in interactive storytelling.
- **Contribution to Educational Technology**: Enhances the field of educational technology by developing a tool that aligns storytelling with pedagogical objectives, supporting personalized and engaging learning experiences.
- **Modularity and Extensibility**: Highlights the modular and extensible nature of the framework, showing its adaptability to various storytelling contexts, educational goals, and future technological advancements.
- **Comprehensive Documentation and Accessibility**: Provides thorough documentation and open-source code, facilitating replication, further research, and practical application by educators, developers, and researchers.

The remainder of this thesis is organized as follows:

- Chapter 2: Literature Review - This chapter reviews relevant literature on LLMs, FSMs, microtask/prompt-chaining and interactive storytelling systems, building the basis for the System Architecture and Design.
- Chapter 3: System Architecture and Design - This chapter provides an in-depth look at the architecture and design of CreatAIlity, including the FSM design and LLM-Prompt-*Incipits* integration.
- Chapter 4: Implementation and Evaluation - This chapter outlines the development process of CreatAIlity, detailing the technical aspects, challenges encountered, and solutions implemented during the creation of the framework. Also, it describes the methodology for evaluating CreatAIlity, including metrics, and analysis of the results.
- Chapter 5: Conclusions - This chapter summarizes the key findings, highlights the achievements of the study, and provides future research directions.
- Appendices - The appendices include supplementary materials such as user testing protocols, sample prompts and outputs, technical documentation, and additional resources relevant to the study.

Chapter 2

Literature Review

2.1 Large Language Models (LLM)

Large language models (LLMs) have transformed natural language processing (NLP), enabling applications like automated content generation and conversational agents. Models like GPT-3 and GPT-4, pretrained on vast datasets, excel in tasks such as translation, question answering, and creative writing. Their ability to generate coherent and contextually relevant text has opened up new possibilities in various domains, including education and creative industries (Lin, 2024; Wu, Terry, and Cai, 2022).

LLMs have significant potential in enhancing educational tools, particularly in storytelling, a critical aspect of early childhood education that promotes creativity, language development, and cognitive skills (Nack and Gordon, 2016). Traditional storytelling methods can be augmented with interactive technologies powered by LLMs, which generate engaging and contextually appropriate stories, adapting to children's interests and responses for a personalized learning experience. This capability has the potential to revolutionize educational content delivery, making it more engaging and effective.

LLMs are also being showed valuable in collaborative creative work. Projects like CoAuthor (Lee, Liang, and Yang, 2022) and Dramatron (Mirowski et al., 2023) demonstrate how LLMs can assist human writers by generating initial drafts, providing prompts, and co-writing long-form texts such as screenplays. These models help overcome writer's block and enhance the creative process by suggesting diverse narrative directions, making them useful tools for authors, screenwriters, and content creators.

On the other hand, these tools are not concerned with following a structured process. When this is the case, the inherent non-determinism large language models exhibit (Ouyang et al., 2023), must be taken into account. This non-determinism means they can produce different outputs even when given the same input prompt.

Part of the stochastic nature of LLMs stems from their design to predict the probability of the next word or token based on the context provided by preceding text. This probabilistic approach is integral to their ability to generate human-like text but also introduces randomness into the output by being influenced by probabilistic sampling methods such as top-k sampling or nucleus sampling (Ouyang et al., 2023). In practical terms, this means that running the same prompt through an LLM multiple times can result in a variety of responses, each potentially differing in both content and quality.

But there is also a variability factor being introduced by hardware processes and even by the model design (Riach, 2019; *OpenAI Platform* n.d.). As a result, identical instructions can yield diverse and unpredictable responses, which poses significant challenges in applications requiring consistency and reliability.

Empirical studies on LLMs, such as those conducted on ChatGPT for code generation tasks, have highlighted the extent of this non-determinism. For example, an analysis of ChatGPT's performance across multiple benchmarks revealed high degrees of non-determinism, with significant variations in the outputs generated from identical prompts. The study found that setting the temperature parameter to zero, which is commonly believed to reduce randomness, does not eliminate this effect entirely. This underscores the challenge of achieving consistent behavior from LLMs: even under controlled conditions, complete determinism is not achievable. (Ouyang et al., 2023)

This non-determinism is clearly and particularly problematic in contexts where consistent and reliable steps and outputs are crucial. For instance, in educational tools aimed at storytelling, the inconsistency of LLM outputs can disrupt the narrative building flow, making it difficult to maintain a coherent story building structure. This unpredictability can confuse users, especially children, who rely on stable and logical progression in educational content.

In the context of developing educational storytelling tools, such as "CreatAIlity", which aim to foster children's narrative creativity while ensuring coherence, understanding and managing the non-determinism of LLMs is crucial.

To mitigate the effects of non-determinism, literature shows some approaches such as using smaller, more focused prompts have shown promise[CITE]. By narrowing the scope of a prompt, the range of possible outputs can be constrained, leading to more relevant and coherent output generation.

Additionally, implementing a hierarchical structure that guides the narrative building flow can help mitigate the unpredictability of LLM outputs, ensuring that the process of coauthoring stories remain coherent and educationally valuable. This structured approach has the potential to not only address the non-determinism but also to enhance the user experience by providing a reliable and engaging storytelling framework.

2.2 Microtasks and Prompt Chaining

Microtasking has been shown to significantly enhance productivity by breaking down larger tasks into smaller, manageable units. This approach has been successfully applied in various domains, including humor generation and writing, where tasks are decomposed into microtasks to streamline the creative process and manage cognitive load (Teevan et al., 2016). The notion of chaining microtasks, as discussed by Cai, Iqbal, and Teevan, 2016, emphasizes the impact of task order on efficiency and continuity, further supporting the structured progression in complex workflows.

Cai, Iqbal, and Teevan, 2016 highlighted the importance of task ordering in microtask chains, demonstrating how the sequence and complexity of tasks influence overall productivity and cognitive load. These findings underscore the relevance of carefully structuring microtasks within FSM states to optimize the storytelling process and maintain narrative coherence.

Task decomposition is a crucial aspect of microproductivity, enabling individuals to complete complex tasks through a series of smaller, well-defined steps. The structured approach of microtasking, ensures that each microtask contributes meaningfully to the overall task while maintaining coherence and consistency throughout the whole process.

The HumorTools (Chilton, Landay, and Weld, 2016) system exemplifies the application of microtasking in creative processes. By dividing humor writing into discrete microtasks, such as identifying aspects, generating reactions, and creating associations, HumorTools provides a structured yet flexible framework that enhances creativity and productivity. The system's dynamic workflow, inspired by design principles, allows tasks to be applied based on context and opportunity, which aligns with the goals and aims of CreatAIlity.

Moreover, the concept of chaining multiple LLM prompts together, as explored in the paper "AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts" (Wu, Terry, and Cai, 2022), which explores innovative methods to improve human-AI collaboration using large language models using these borrowed concepts. It addresses key challenges such as the lack of transparency, controllability, and coherence in LLM outputs, especially for complex tasks, discussing some similar issues faced in this work. As a result, this research aligns closely with the objectives of the CreatAIlity framework:

- Chaining LLM Prompts: Introduces the concept of chaining multiple LLM prompts together, where the output of one step becomes the input for the next. This approach helps in breaking down complex tasks into manageable subtasks, improving task outcomes by ensuring each sub-task is handled effectively by the LLM.
- Transparency and Controllability: By visualizing the chain of prompts and allowing users to modify and interact with each step, the system enhances transparency and gives users greater control over the LLM's outputs. This method helps users understand how the AI arrives at its results, making the interaction more predictable and user-friendly.
- Improved Collaboration: The chaining approach encourages a more collaborative interaction between humans and AI, where users can iteratively refine and improve the AI-generated content. This process fosters a sense of partnership and shared authorship.
- Modular Design: The modular design of chaining LLM prompts allows for flexible customization and adaptation of the AI to various sub-tasks, enhancing its applicability across different domains and tasks.

2.3 Human-AI Collaboration in Creative Writing and *Incipits*

Since the 1990s, storytelling has garnered significant interest within the Human-Computer Interaction (HCI), Interaction Design and Children (IDC), and Artificial Intelligence (AI) communities. Researchers have explored how interactivity can enhance engagement, enjoyment, and fun in storytelling, thereby fostering new avenues for children's creativity. These efforts aim to amplify the educational benefits of traditional storytelling methods by integrating interactive elements that make the experience more dynamic and engaging for young audiences (Garzotto, Paolini, and Sabiescu, 2010).

The landscape of Human-AI collaboration in creative writing has significantly evolved with the advent of large language models (LLMs) like GPT-3, GPT-4, and others. These models have been used in various creative domains, including scriptwriting, storytelling, and collaborative fiction. Projects such as CoAuthor and Dramatron illustrate how LLMs can assist human writers by generating initial drafts, providing prompts, and even co-writing long-form texts like screenplays and theatre scripts (Lu et al., 2024; Mirowski et al., 2023; Lee, Liang, and Yang, 2022).

CoAuthor focuses on creating a dataset designed for human-AI collaborative writing, capturing interactions between writers and GPT-3 across various writing sessions. The dataset helps in understanding the generative capabilities of LLMs, their influence on human writing, and the collaborative dynamics between humans and AI. Researchers found that LLMs could provide new ideas and enhance the creative process, but also noted challenges in ownership and satisfaction among writers (Lee, Liang, and Yang, 2022).

Dramatron, an interactive co-writing tool that uses hierarchical story generation techniques to assist in writing scripts and screenplays. By using structured prompts and generating text hierarchically, Dramatron helps maintain coherence over long narratives, addressing one of the key limitations of flat text generation by LLMs. The project involved industry professionals who provided feedback that helped refine the tool, emphasizing the importance of domain-specific expertise in developing such systems (Mirowski et al., 2023).

The integration of large language models (LLMs) into creative writing has opened new possibilities for enhancing narrative construction and storytelling. One significant advancement in this area is the use of *incipits* (Addone, Palmieri, and Pellegrino, 2022), or story starters, which provide a structured yet flexible framework to guide writers through the narrative creation process. *Incipits* serve as scaffolding resources that support both educators and learners by offering initial prompts that can spark creativity and guide the development of stories. This method has been shown to significantly enhance children's engagement and creativity by providing a clear starting point while allowing for individual or collaborative story development.

LLMs have revolutionized creative writing by offering various capabilities such as text summarization, paraphrasing, elaboration, dialogue generation, and story seeding (Gmeiner and Yildirim, 2023). These capabilities can be utilized in conjunction with *incipits* to create a robust framework for narrative development. For example, an LLM can generate initial story ideas based on a given *incipit*, which children can then expand and personalize. This collaborative process not only fosters creativity but also supports language development and cognitive skills.

The combination of LLMs and *incipits* creates a collaborative framework that ensures coherence and consistency in generated stories. This integration allows for a guided storytelling process where each part of the story adheres to a coherent structure, while still allowing for creative input from the children. This structured approach helps maintain logical flow and supports children's narrative creativity by providing a guided yet flexible framework that adapts to their input and keeps the story development on track.

Interactive storytelling tools, such as those described in the literature, support educators and learners in inventing and authoring stories. These tools provide digital learning environments where children can start with default templates or teacher-defined *incipits* and continue the story using a clear and concise interface. Such tools often include features like image libraries and word clouds to inspire creativity and help overcome writer's block (Addone, Palmieri, and Pellegrino, 2022). Some of these features can even be provided by the LLM, without the need for additional material or input other than the user interaction and prompt design.

By leveraging the structured approach of *incipits* and the generative capabilities of LLMs, storytelling tools can offer a structured yet flexible framework that supports narrative creativity while ensuring coherence and consistency in the generated stories. The combination of digital tools and traditional storytelling methods offers new opportunities for children to explore their creativity and develop their narrative skills.

This approach not only enhances the educational value of storytelling tools but also engages children in the creative writing process, fostering their language development, cognitive skills, and socio-emotional growth. By integrating LLMs and *incipits* into these tools, educators can provide a more engaging and effective storytelling experience that supports children's overall development.

In conclusion, this approach provides a reliable and engaging storytelling experience that fosters creativity and supports educational objectives. By enhancing the learning experience for children, these tools have the potential to help them prepare for future academic and personal success.

2.4 Finite State Machines with Stack-Based Control

Finite State Machines (FSMs) are mathematical models of computation that are used to design both computer programs and sequential logic circuits. They consist of a finite number of states, transitions between these states, and actions, which are triggered by inputs and events. FSMs are characterized by their ability to be in exactly one of a finite number of states at any given time. The system transitions from one state to another in response to external inputs, and the transitions are governed by a set of defined rules.

The main characteristics of FSMs include their deterministic nature, where the next state of the machine is determined by the current state and the input. This determinism ensures predictability and consistency, which are crucial for applications requiring structured and coherent outputs. FSMs can be visualized using state diagrams, which provide a clear and intuitive representation of the states and transitions, making it easier to design and debug complex systems (Alagar and Periyasamy, 2011; Ehrlinger and Wöß, 2016) and also facilitate building user-friendly visual tools to implement them.

FSMs are particularly suitable for managing part of the non-determinism inherent in large language models (LLMs) in the context of structured processes, more specifically collaborative story building. LLMs, while powerful in generating diverse and creative text, often produce outputs that lack coherence and consistency. As noted before, this non-determinism can be problematic in educational contexts, where structure and reliable content is essential (Liu et al., 2024). By integrating

FSMs with LLMs, a structured framework can be established that guides the narrative flow, ensuring that the story-building process remains coherent and follows a logical sequence.

Hierarchical Finite State Machines enhance the traditional FSM by introducing a hierarchical structure that organizes states into multiple levels of abstraction. Each HFSM consists of high-level states, which can encapsulate lower-level sub-states, enabling a top-down decomposition of complex control processes. This structure supports modularity and reusability, allowing components to be independently designed and debugged. HFSMs use a stack-based control mechanism, where the current state is pushed onto a stack during state transitions, facilitating the management of nested and recursive state calls. This approach ensures a seamless return to the previous state after an interruption. Additionally, HFSMs are event-driven, triggering state transitions based on specific events or conditions, which maintains the logical flow and coherence of processes. Their ability to handle dynamic changes and interruptions, coupled with their modular and reusable design, makes HFSMs highly flexible and extensible for various applications (Sklyarov, 1999; Alagar and Periyasamy, 2011).

Event-driven transitions ensure that state changes occur in response to relevant inputs, maintaining the logical flow of the process. In the context of storytelling, events such as user choices, generated prompts, or predefined conditions trigger transitions between different story phases and sub-states. This event-driven approach ensures that the narrative progresses logically and coherently, supporting the user's understanding and engagement. By providing a structured yet flexible framework, FSM helps users develop their narrative creativity while adhering to educational objectives.

However, HFSMs can introduce unnecessary complexities for certain applications, such as educational storytelling, where a simpler and more streamlined approach may be more effective. The hierarchical nature and extensive control mechanisms of HFSMs, while beneficial for highly complex systems, can add layers of abstraction that may not be needed for guiding narrative development in a structured yet flexible manner. Nonetheless, some concepts addressing FSM limitations can be borrowed to improve FSMs for specific uses.

The modular and reusable design of the FSM enhances extensibility and flexibility. Individual states and transitions can be designed as independent modules, facilitating their reuse in different contexts, tools, or processes. This modularity allows developers to easily extend or modify the framework to accommodate new elements or goals. For example, new story phases or sub-states can be added without disrupting the existing structure, ensuring that the tool remains adaptable to evolving needs. This flexibility supports continuous improvement and customization, ensuring that it remains an effective and engaging educational tool.

Chapter 3

System Architecture and Design

Building on the concepts and objectives outlined in the previous sections, this section establishes the requirements, constraints, and definitions that guided the design phase of CreatAIlity. Following this, a detailed modeling and description of the implementation of its modules and components is provided.

3.1 Overview of CreatAIlity Architecture

The CreatAIlity framework integrates the structured control of Finite State Machines (FSM) with the creative capabilities of Large Language Models (LLMs) to manage narrative building. The FSM provides a backbone for the narrative structure, ensuring logical progression and educational coherence, while LLMs generate creative content. By chaining LLM prompts into microtasks, the system maintains coherence and manages complexity, enhancing storytelling capabilities for children.

FSMs ensure the logical and educational flow of the narrative, adhering to predefined goals and arcs. LLMs add creativity and variability, engaging children and supporting their narrative creativity. This dual approach balances creativity with structure, providing a coherent and educational storytelling experience (Alagar and Periyasamy, 2011; Mishra et al., 2019; Yang and Tiddi, 2020).

In this architecture, information handling is crucial. Each state or step uses and provides information, creating a dependency chain. The Resource-Based FSM approach defines states by the resources they need and provide, ensuring states execute only when all required resources are available. This promotes a coherent progression through the narrative.

A stack-based control mechanism manages state transitions, handling nested and recursive state calls effectively. This ensures continuity, allowing the system to return seamlessly to previous states after completing nested states. The system iteratively searches for states that can provide the necessary resources, pushing states into the execution stack until the final resource provider is reached.

Microtasks and prompt chaining break down the narrative generation process into manageable tasks, ensuring focus and coherence. Each microtask, associated with a state machine step, handles specific aspects of the story, such as setting descriptions, character introductions, or plot developments. These tasks are linked in a chain, where each task's output serves as the input for the next task. This method ensures that the narrative remains structured while allowing for creative input at each stage.

Incipits are materialized by both the initial prompting given by the application designer for a state and the LLM suggestions during the chat, influenced by the given prompt. This approach guides narrative creation, maintaining structure while allowing creative input from both the LLM and the child. The final content is summarized and extracted from outputs using techniques such as prompt instructions and regexes. These outputs then become resources, stored in the state machine's memory and usable by other states.

In a nutshell, by combining a stack-controlled, resource-based FSM with prompt chaining, the framework mitigates the unpredictability of LLM outputs by focusing each task in a more atomic format. Each FSM state represents a story part, such as the introduction, conflict, or resolution, with transitions triggered by resource dependencies generated by user interactions and choices. These dependencies are extracted and fed into subsequent states as needed. This structured approach ensures that the overall story remains coherent and aligned with educational objectives, effectively integrating advanced NLP technologies into practical educational tools (Ehrlinger and Wöß, 2016; Mishra et al., 2019).

3.2 Design of the CreatAIlity Framework for Storytelling

The conceptualization of CreatAIlity encompasses a wide array of modules and functionalities. Central to the design of this framework are the concepts of prompt chaining, microtasks, and *incipits*, which play a pivotal role in shaping the tool's architecture. These elements, along with their specific characteristics and goals, must be carefully considered when designing the framework.

Additionally, modules that facilitate seamless user interaction, control and execution, application state persistence, resource management, and LLM integration are fundamental to CreatAIlity's core functionality. Furthermore, robust debugging tools should be incorporated to provide developers with enhanced visibility and control during application development. An i* (Pimentel et al., 2019) model illustrating these concepts is presented in 3.1.

3.2.1 Integration with LLMs and Prompt Chaining

To enhance creativity and manage LLM generative capabilities, the CreatAIlity framework integrates prompt chaining and microtasking techniques into its design. These approaches break down the narrative generation process into smaller, manageable tasks, each handled by the LLM with focused prompts. This ensures coherent and creative outputs while maintaining control over the narrative flow.

Microtasks and Prompt Chaining

Microtasking breaks down narrative generation into smaller tasks, each handling a specific narrative element. These tasks are chained together, with each task's output serving as the input for the next task. This structured approach enhances coherence and consistency in the storytelling process and aims to motivate application developers to break-down the process into smaller parts.

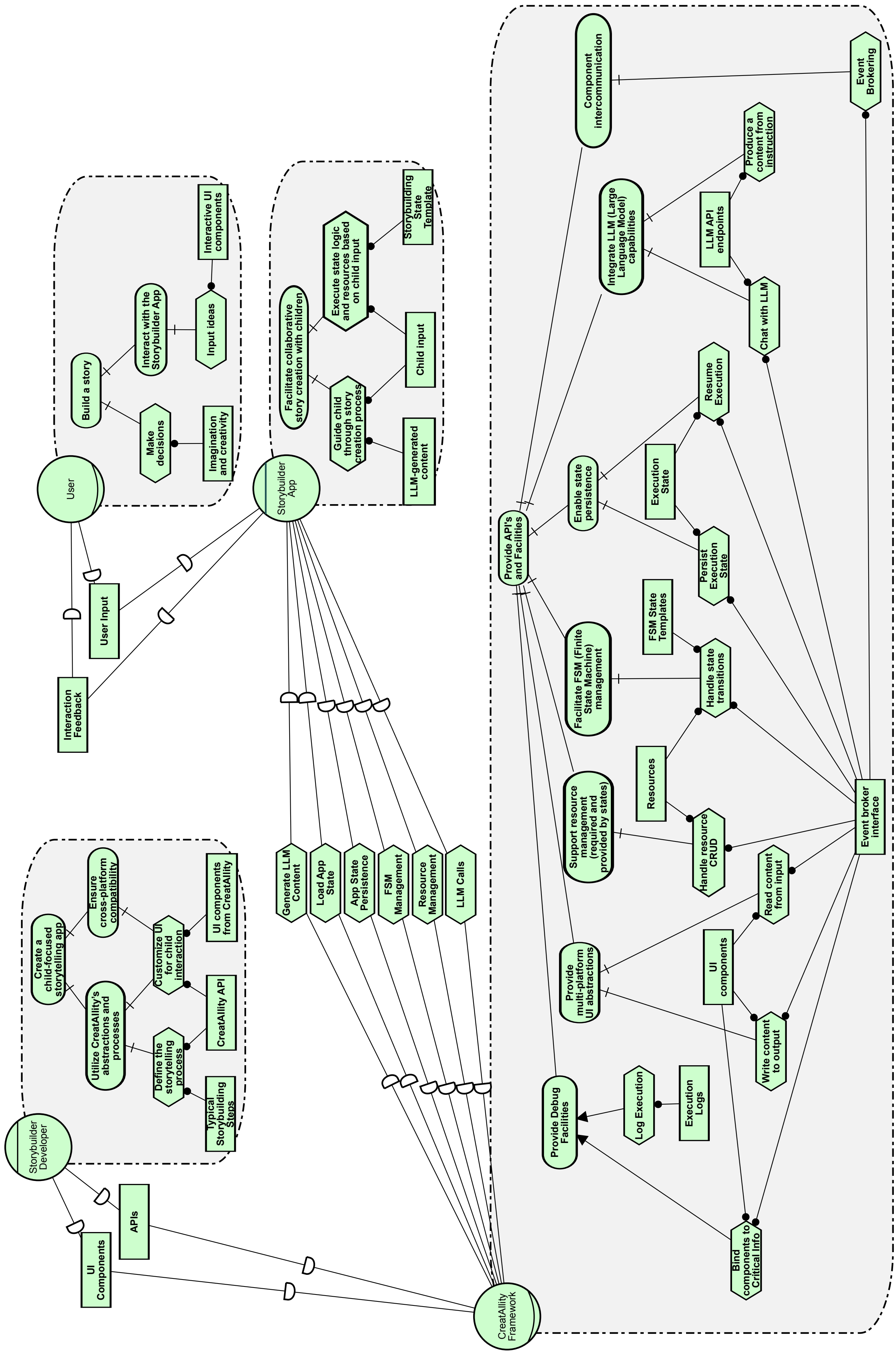

FIGURE 3.1: CreatAIlity i* model (Dependencies kept on Actors for clarity)

Utilizing *Incipits*

Incipits are the initial prompts that guide Large Language Models (LLMs) in generating relevant content. These prompts also serve as *incipits* during user interactions with the LLM, helping to initiate and structure the conversation. Careful design of these initial prompts is crucial to ensure that the LLM offers appropriate suggestions throughout the dialogue. Additionally, it is possible to instruct the LLM to generate explicit *incipits* based on user input or previously generated content. For example, after the user defines a theme, the LLM could be directed to provide a partial starting sentence to help the user begin their narrative.

Structured *incipits* play a key role in ensuring that each microtask remains focused, aligning the generated content with the overall structure of the story. Moreover, *incipits* can be used as suggestions provided by the LLM during the conversation to facilitate storytelling, helping to maintain coherence and direction throughout the narrative.

Prompt Chaining

Prompt chaining in the CreatAIlity framework enhances the storytelling process by ensuring that each generated output logically and creatively builds upon the previous one. By breaking down the narrative into smaller, manageable microtasks, each associated with specific prompts or *incipits*, the system can guide the LLM to produce focused and coherent content at every step. The resource-based strategy ensures that each step is supplied with all necessary resources for its prompting and processing logic.

This approach minimizes the risk of off-topic or inconsistent outputs, as each prompt is carefully designed to align with the overarching story structure. Prompt chaining allows for dynamic adaptation and customization, making it easier to incorporate user interactions and developments while maintaining a continuous and cohesive narrative flow.

Modular and Flexible Design

The modular design of chaining LLM prompts enables flexible customization and adaptation. Each FSM state can be linked to specific *incipits* and microtasks, streamlining complex storytelling workflows, ensuring relevant and coherent LLM outputs, and guiding the user to provide targeted inputs.

3.2.2 Resource-based State Management

The essence of the Resource-based FSM approach lies in defining states by the resources they require and provide. These resources, whether content, flags, or any data representable in memory, are essential for state execution and are produced as outcomes of that execution. This method ensures logical and coherent narrative progression by allowing transitions between states only when the necessary resources are available.

Resource Definition

In CreatAIlity's FSM, resources can include content such as character introductions, setting descriptions, plot resolutions, or completion flags for hierarchical grouping of states that contribute to a larger step. Larger logical steps can be grouped, such as "Setting Description" and "Character Introduction," which can both be required by a parent state "Introduction." This parent state binds them together and represents the completion of a larger state. This setup facilitates dependency relations so that other states can depend on the parent state instead of each individual child state.

By definition, each state specifies:

- **Required Resources**: Multiple prerequisites needed for state execution.
- **Provided Resource**: One Output generated by the state for subsequent states.

While it is possible to provide multiple resources, this goes against the atomicity aimed for by using microtasks. Limiting the provided resource to one encourages the application designer to focus on defining the smallest necessary part and composing them effectively.

State Transitions

State transitions depend on the availability of required resources. When a state completes, it provides resources that are stored in a shared memory. The FSM then checks if the next states can execute based on the available resources, ensuring a structured narrative flow. This method guarantees that each state transition is logical and that the narrative progresses coherently.

It is up to the application developer to define and implement the best way to determine when a state has been completed. Callbacks and events allow the client application to access relevant information, process it, and signal that it has achieved its goal. This mechanism can return actions such as dynamically stacking a new state, as well as creating, updating, or even removing a given resource.

3.2.3 Stack-based Control Mechanism

To manage the desired sequential interactions derived by each state required and provided resources, the Resource-based FSM employs a stack-based control mechanism. This approach handles nested and recursive state calls seamlessly, maintaining storytelling continuity. By pushing states onto the stack and returning to previous states after nested ones are completed, the system ensures a smooth and coherent flow even during complex interactions.

State Stack

The state stack keeps track of states required by the current state (starting with the initial state) for resource requirements that are not yet available, stacking their providers and waiting for their completion. When a state execution starts, the FSM checks if it has all the resources required for execution. If not, the current state is pushed onto the stack, and the FSM searches for a state that can provide each missing requirement to become the new current state.

This process is repeated for each new resource dependency-derived tree from the current state requirements, allowing the system to revert to previous states once their requirements are met. After completing all necessary stacked states and iteratively satisfying all requirements, the system resumes the previous state by popping it from the stack. This mechanism ensures that the narrative flow remains continuous and contextually accurate.

3.3 Framework Modules Description and Specification

Here the framework modules are described and specified. In figures 3.2, 3.3 and 3.4, we have an overview of the main modules and their intercommunication.

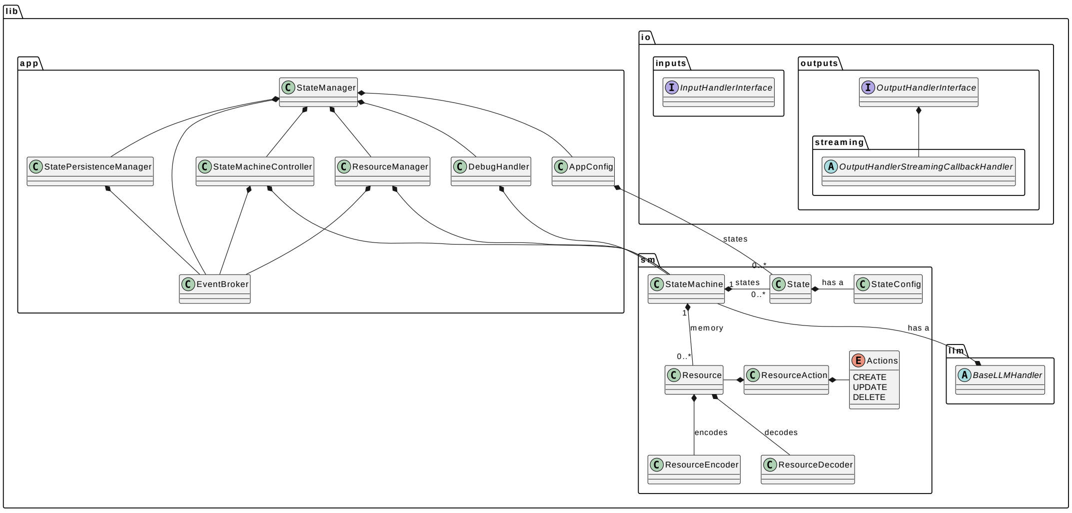

FIGURE 3.2: Core Modules and Abstractions

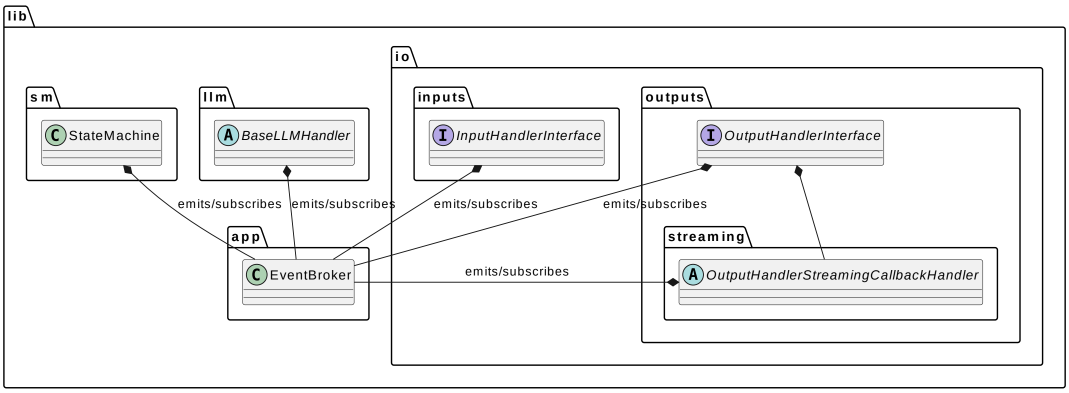

FIGURE 3.3: Event Broker Module Connections

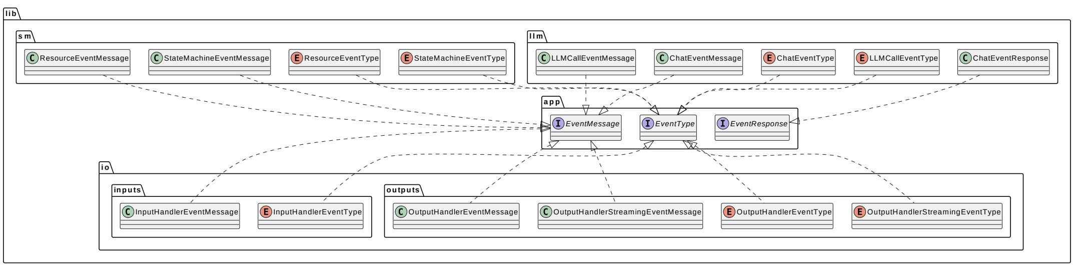

FIGURE 3.4: Event Types, Messages and Responses

3.3.1 Core Library

Input-Output Handling Module

Motivation: Different interfaces (console, GUI, web) require specialized methods for capturing user input and displaying outputs. Abstracting input and output handling allows the application to remain agnostic of the input source, enhancing flexibility and extensibility. This abstraction also facilitates adaptation to different use cases, extending the system's basic capabilities.

Input Components (Figs. 3.5 and 3.6)

- **InputHandlerInterface**: Abstract base class defining methods for handling inputs.
- **ConsoleInputHandler**: Handles input via the console.
- **TkinterInputHandler**: Handles input via a Tkinter GUI.
- **WebSocketInputHandler**: Handles input via WebSocket.

Output Components (Figs. 3.7 and 3.8):

- **OutputHandlerInterface**: Abstract base class defining methods for handling outputs.
- **ConsoleOutputHandler**: Displays output in the console.
- **TkinterOutputHandler**: Displays output in a Tkinter GUI.
- **WebSocketOutputHandler**: Sends output to a web client via WebSocket.
- **StackTkinterOutputHandler**: Displays hierarchical or stacked information in Tkinter.
- **StackWebSocketOutputHandler**: Displays hierarchical or stacked information via WebSocket.
- **OutputHandlerStreamingCallbackHandler**: Handles streaming callbacks from the LLM, updating its child (concrete implementation of the output handler) in real-time.

Requirements:

- **Modularity**: Should encapsulate each input and output method in its own class.
- **Extensibility**: Should allow new input and output methods to be added easily.
- **Reusability**: Should centralize common functionality to reduce code duplication.
- **Consistency**: Should ensure consistent output formatting and behavior across different interfaces.
- **Separation of Concerns**: Should separate output logic from business logic.
- **Customization**: Should allow for customized behavior in different output handlers.

Concerns:

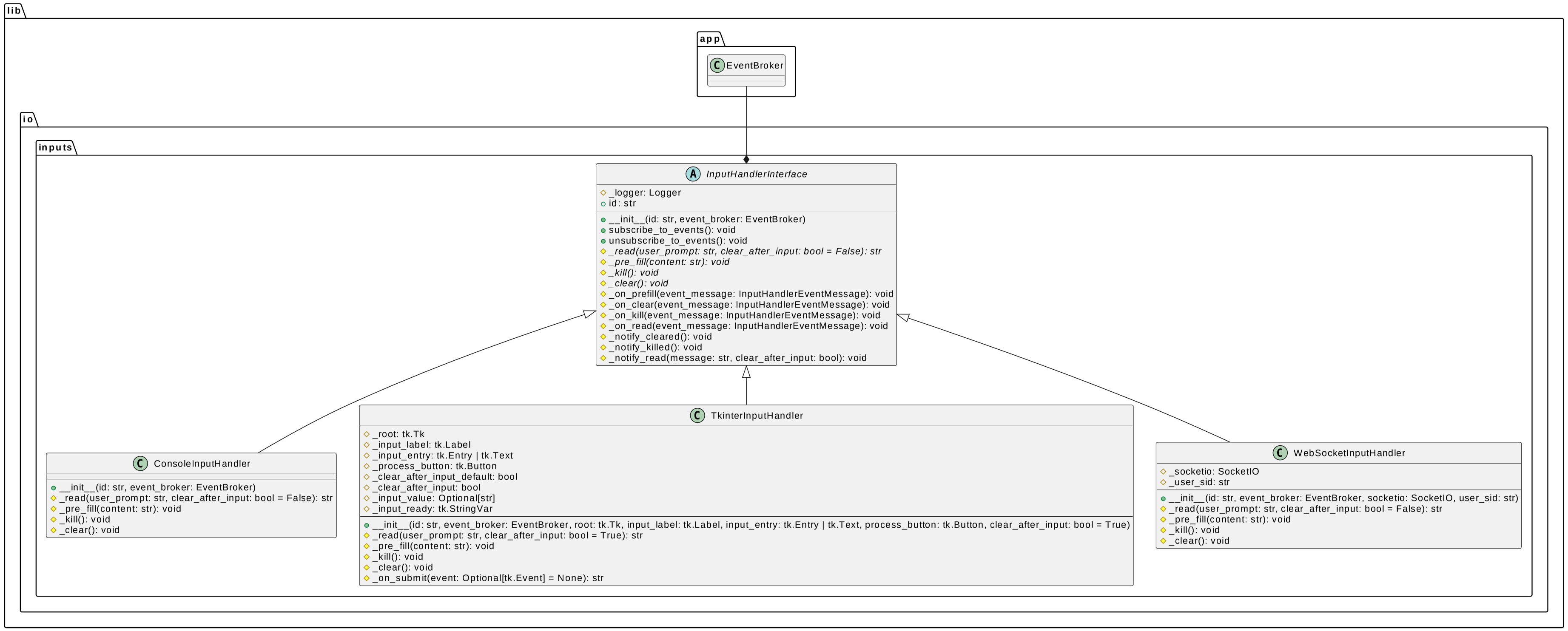

FIGURE 3.5: IO Input Module Class Diagram

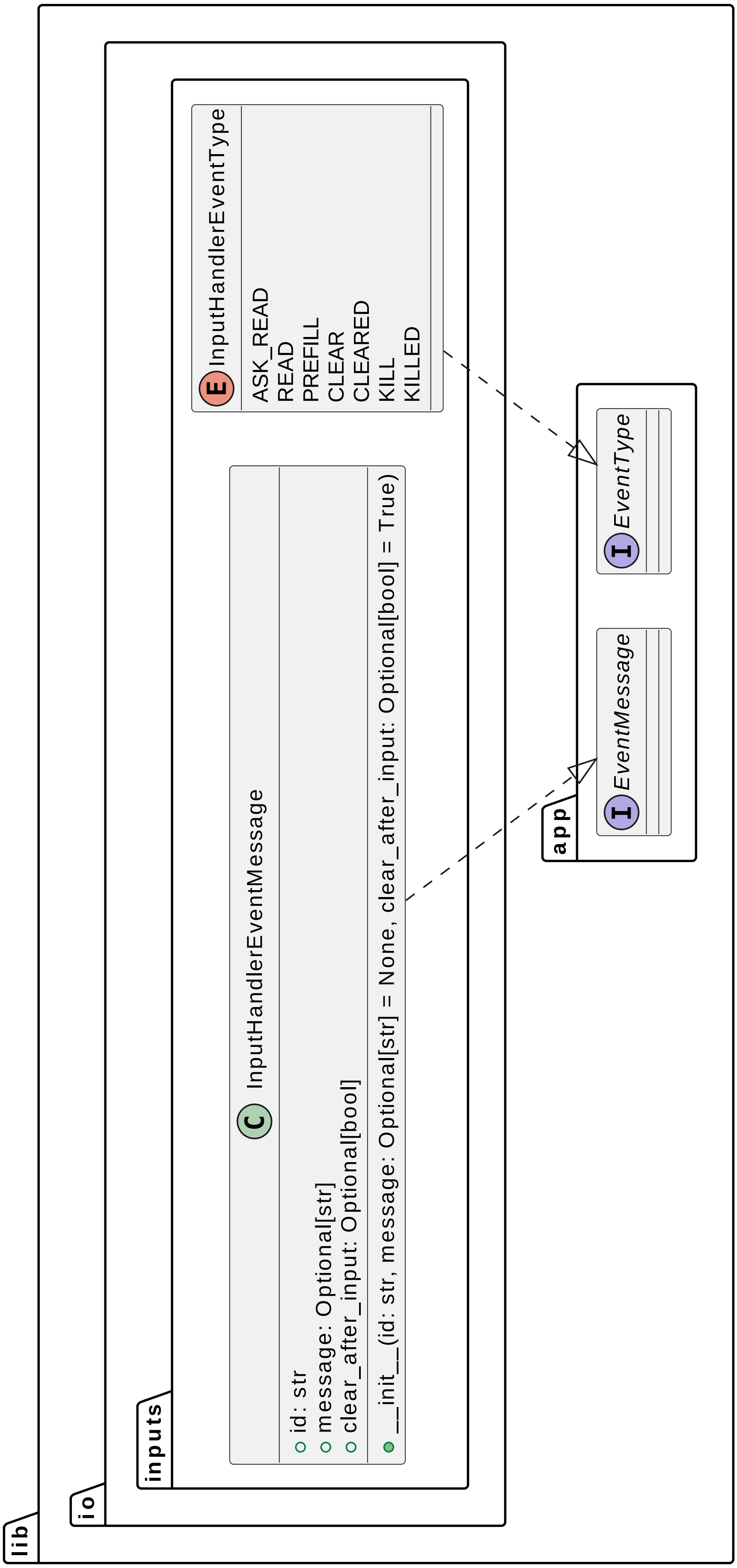

FIGURE 3.6: IO Input Module Events Class Diagram

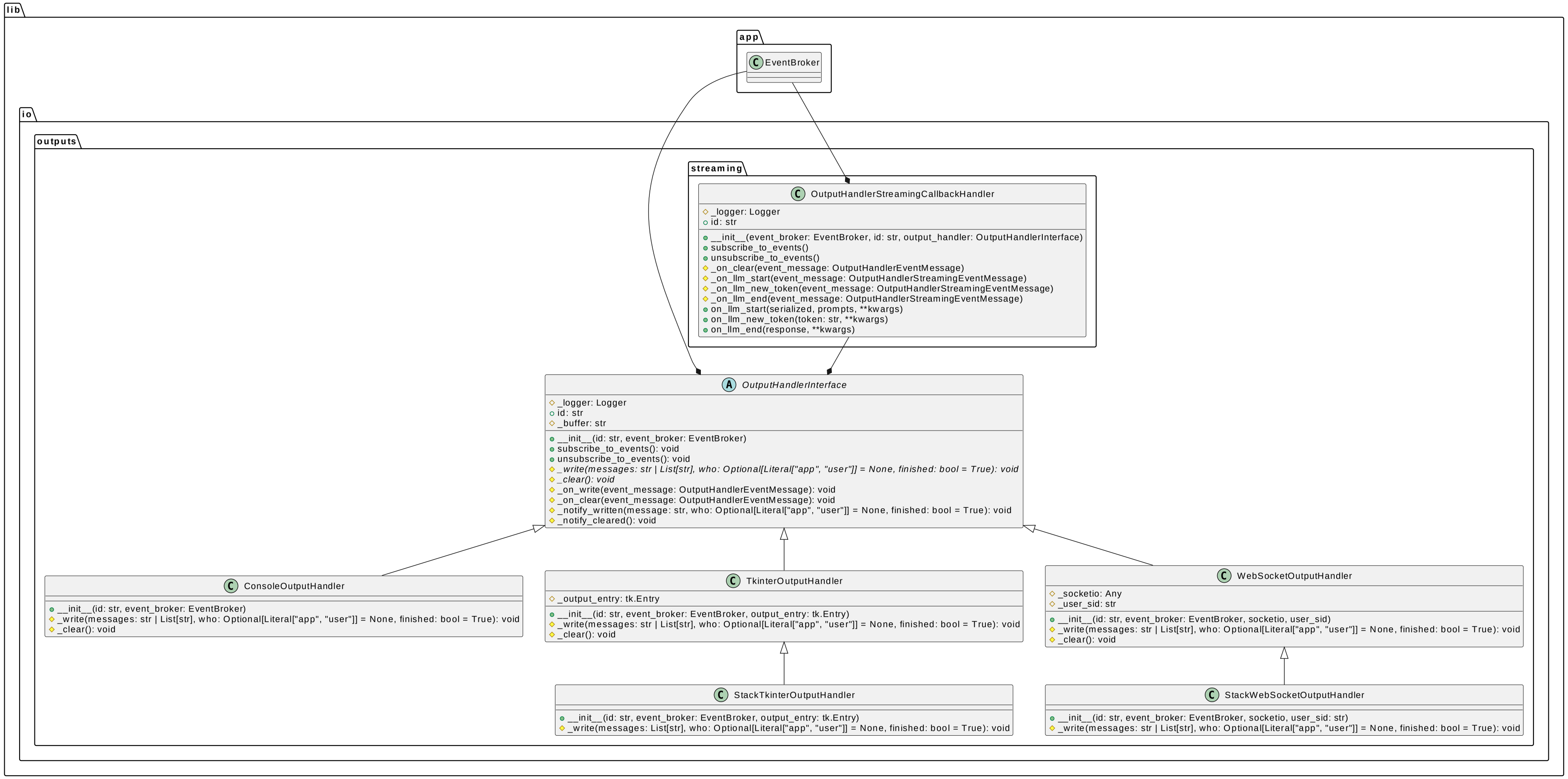

FIGURE 3.7: IO Output Module Class Diagram

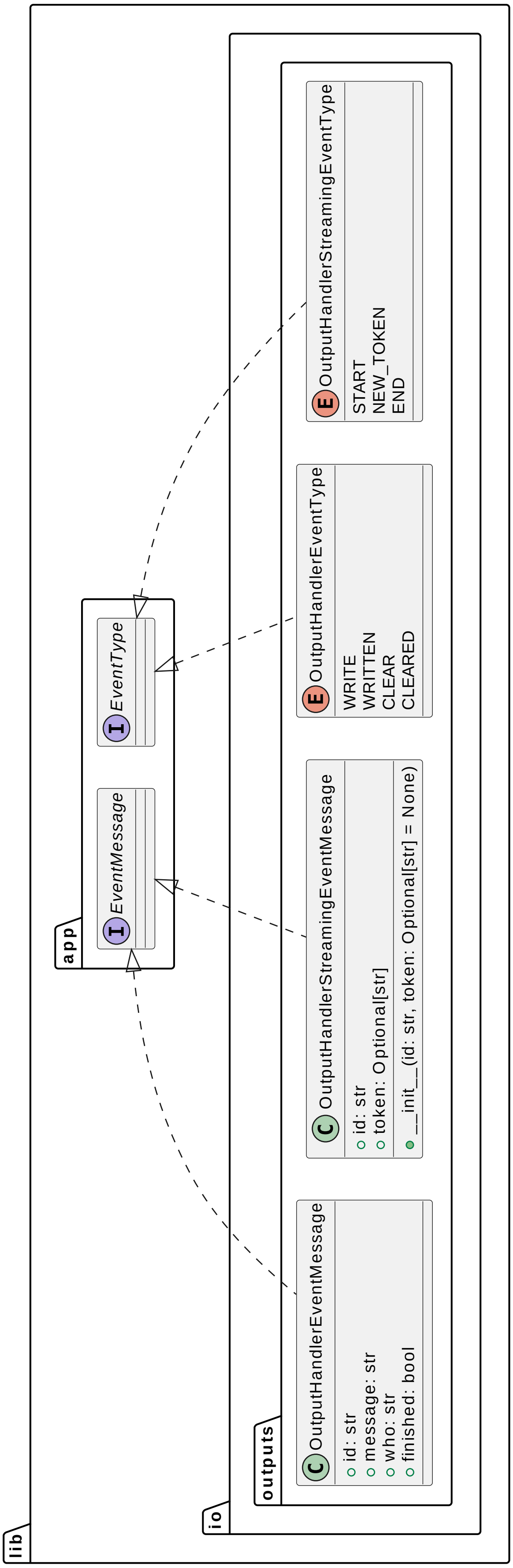

FIGURE 3.8: IO Output Module Events Class Diagram

- **Complexity in Extending**: Adding new input methods may require a deep understanding of both the framework and the specific IO provider.
- **Dependency on Specific Libraries**: The implementations are tightly coupled with libraries like Tkinter and Flask-SocketIO. Streaming output might need adapters for different technology stacks.

LLM Handling Module

Motivation: Interacting with language models involves complex logic for sending prompts and handling responses. Abstracting these interactions simplifies integration and allows for different LLMs to be used interchangeably within the application.

Components (Figs. 3.9 and 3.10):

- **BaseLLMHandler**: Abstract base class for handling interactions with a language model.
- **OllamaHandler**: Manages interactions with the Ollama language model.
- **TGIHandler**: Manages interactions with the Text Generation Inference (TGI) language server from Hugging Face.

Requirements:

- **Abstraction**: Should abstract LLM interactions into handlers to simplify application logic.
- **Extensibility**: Should allow new LLMs to be integrated by implementing the abstract base class.
- **Real-time Updates**: Should support real-time updates via streaming output callback handlers.

Concerns:

- **Complexity in Debugging**: Debugging LLM interactions can be complex and may require specialized tools or techniques.
- **Resource Intensive**: Interacting with LLMs requires significant computational resources, which can be costly, especially in cloud environments.

State Machine Module

Motivation: Managing the flow of an application through different states is essential, particularly in interactive applications. State machines provide a structured approach to handling state transitions and related logic.

Components (Figs. 3.11 and 3.12):

- **StateMachine**: Manages transitions between different states based on application logic and LLM responses.
- **StateConfig, State**: Define the configuration and behavior of individual states.
- **Resource**: Represents individual pieces of data (resources) managed by the state machine.

Requirements:

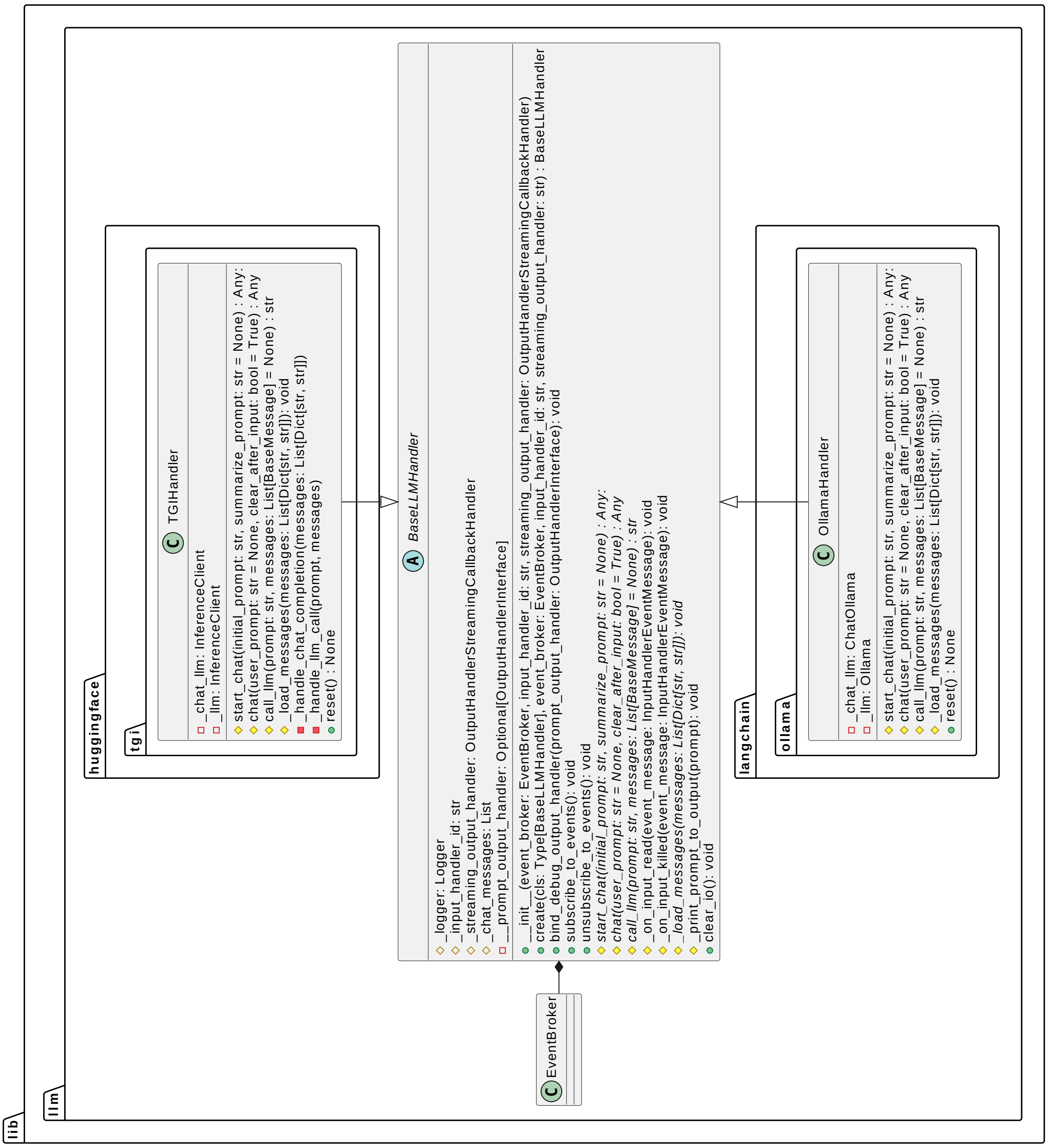

FIGURE 3.9: LLM Module Class Diagram

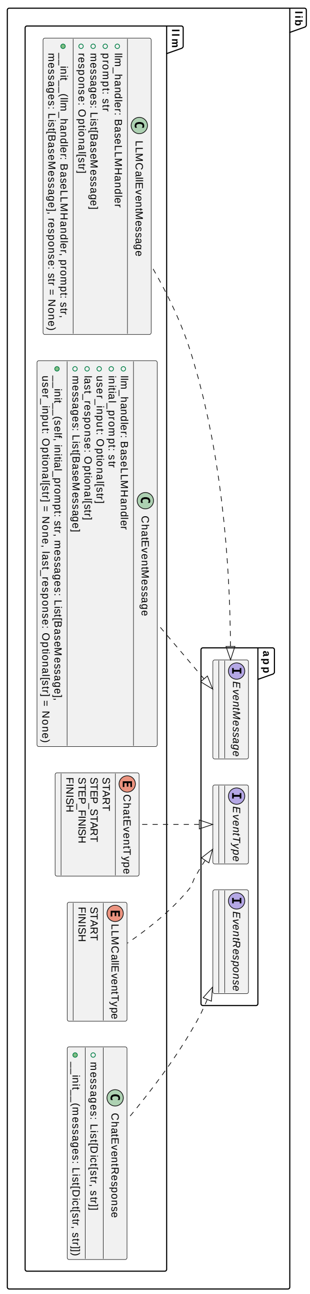

FIGURE 3.10: LLM Module Events Class Diagram

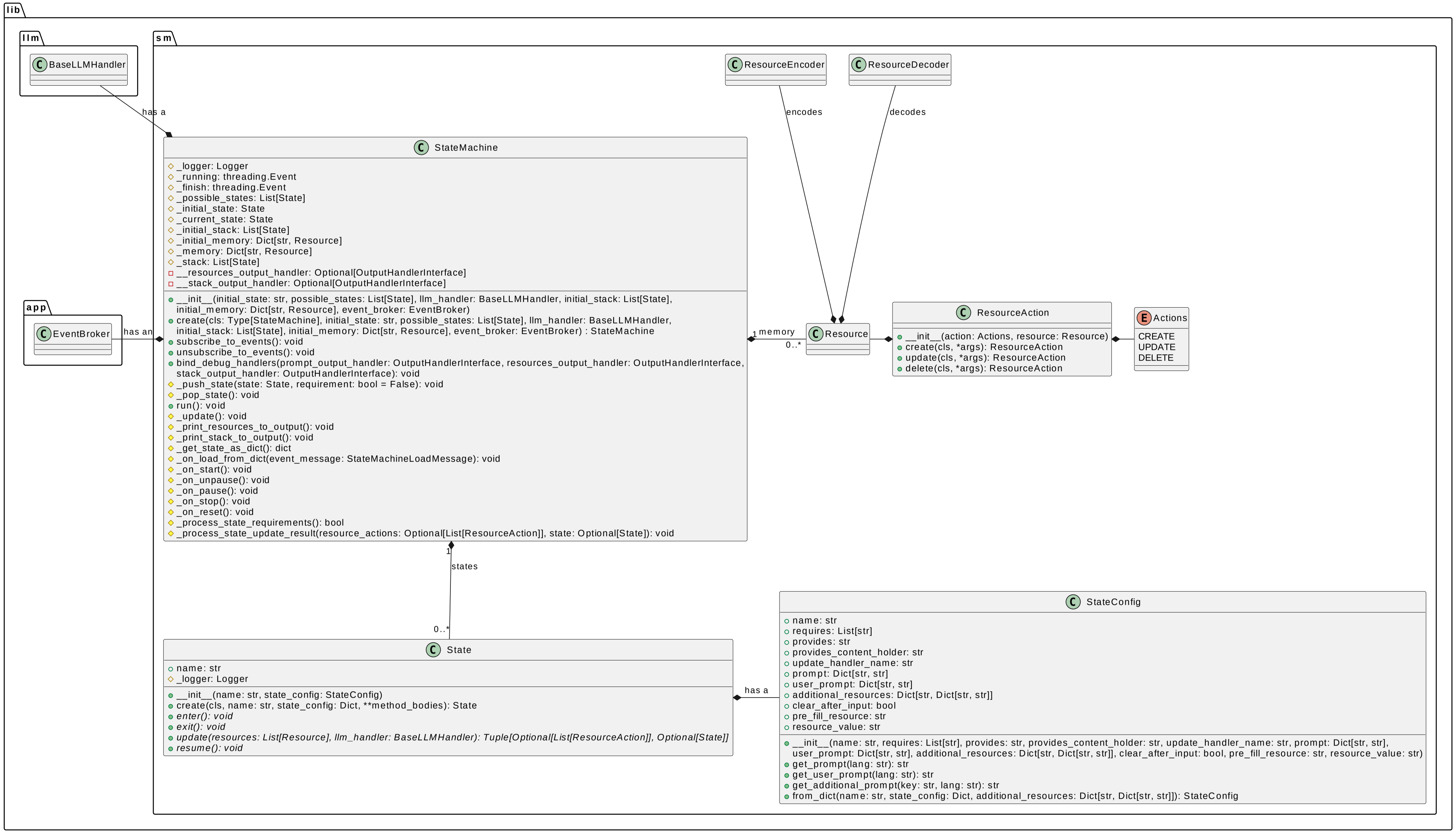

FIGURE 3.11: State Machine Module Class Diagram

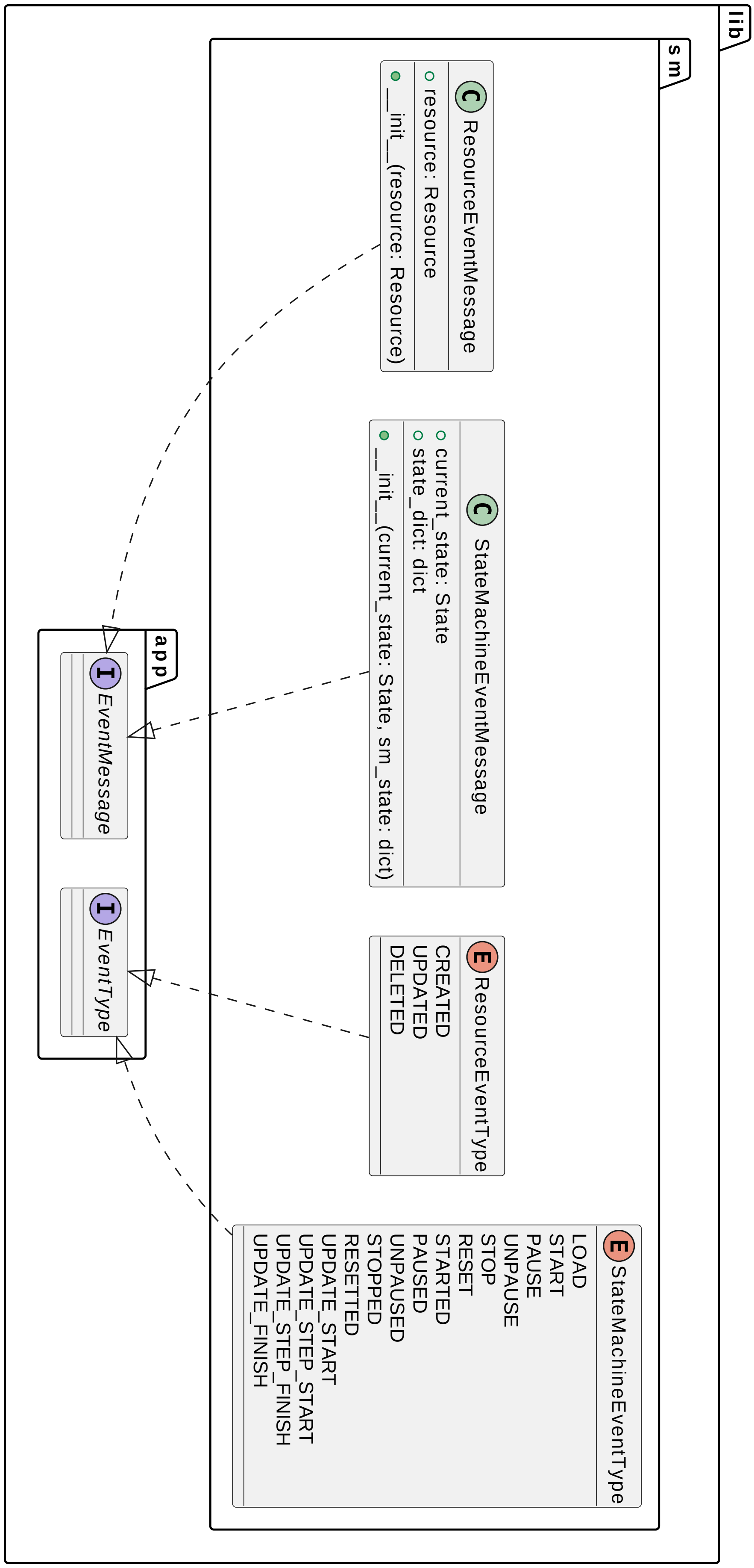

FIGURE 3.12: State Machine Module Events Class Diagram

- **Structured Flow**: Should provide a clear and structured method for managing application flow.
- **Scalability**: Should support easy addition of new states and transitions.
- **Debugging**: Should provide explicit state transitions to facilitate debugging and understanding of the application flow.

Concerns:

- **Complexity**: The state machine can become increasingly complex as the number of states and transitions grows.
- **Scalability**: Adding new states and transitions might introduce unintended side effects.
- **Performance**: Managing state transitions and large state stacks can lead to performance bottlenecks.

Management Module

Motivation: The Management module serves as the central hub for managing the application's state, coordinating various components, and handling interactions between the state machine, LLM handlers, and input/output handlers. Centralizing these responsibilities ensures the application remains organized, modular, and capable of handling complex interactions.

Components (Figs. 3.13 and 3.14): Due to the critical role of these components, they were also especified in greater detail.

- **StateManager**: The core class that coordinates the setup, event subscription, and execution of the state machine and related handlers.
  - **Motivation:** The StateManager class is pivotal in orchestrating the overall operation of the application's state management system. It integrates and configures key components such as the StateMachineController, DebugHandler, ResourceManager, and StatePersistenceManager, ensuring they work in harmony to manage state transitions, resource allocation, and event-driven interactions. By overseeing the initialization and coordination of these modules, the StateManager ensures the application runs smoothly, handling state transitions effectively, maintaining persistence, and responding appropriately to user inputs.
  - **Requirements:**
    - **Centralized Management**: Should centralize the control of the application's state and the interaction between various components.
    - **Extensibility**: Should accommodate the addition of new state machine classes, LLM handlers, and input/output handlers.
    - **Event-Driven Architecture**: Should leverage an event-driven architecture to allow for decoupled communication between components, enhancing modularity.
    - **Customization**: Should support custom handlers and configurations to allow for tailored application behavior.

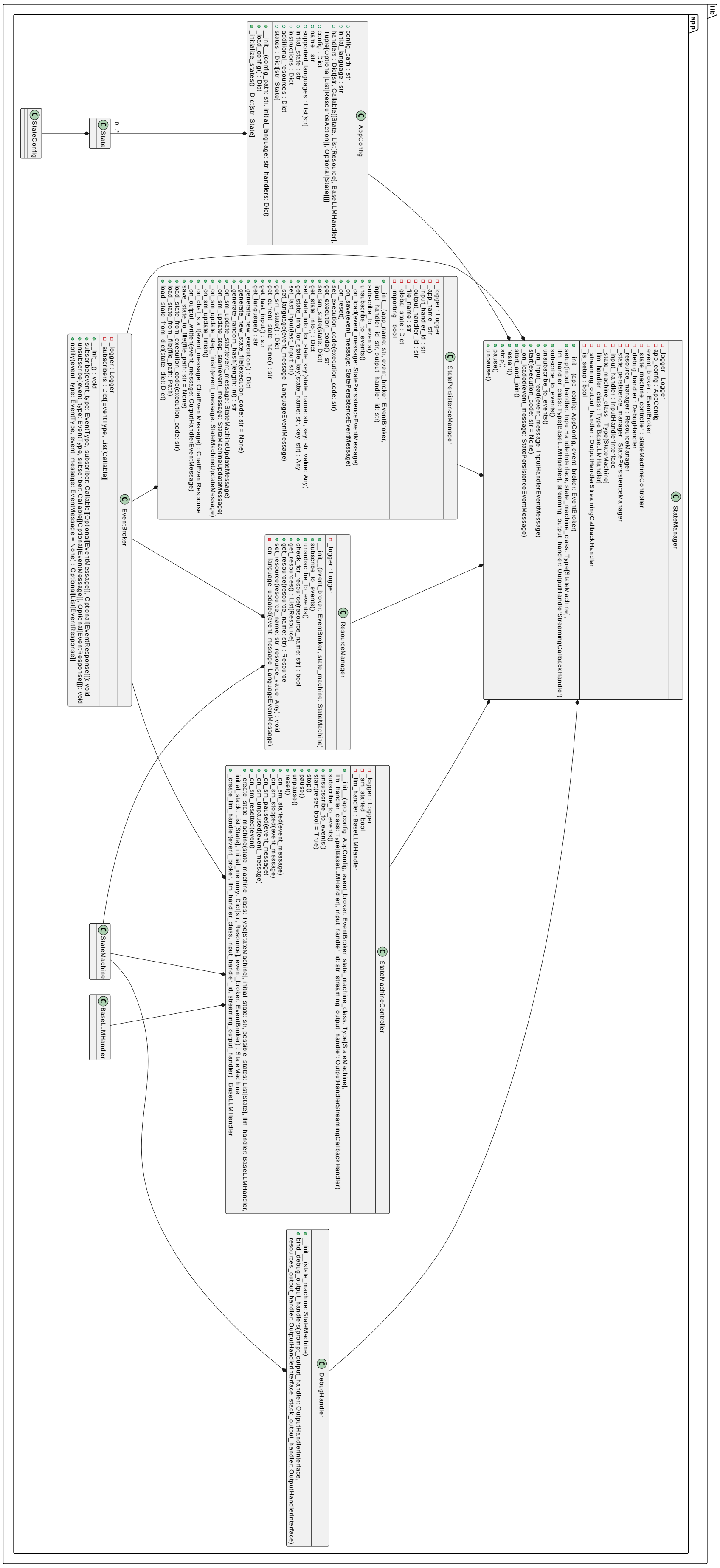

FIGURE 3.13: State Manager Coordinator, Subordinate Managers and Event Broker Class Diagram

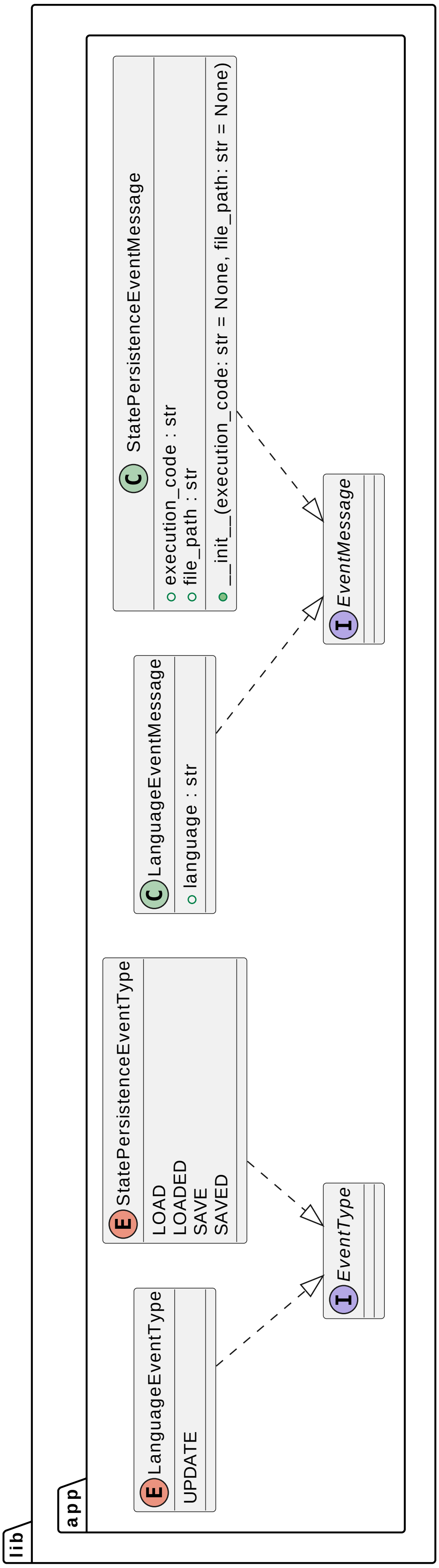

FIGURE 3.14: State Manager Coordinator, Subordinate Managers and Event Broker Events Class Diagram

- **Concerns:**
  * **Setup Complexity**: Properly configuring the StateManager requires understanding various components and their interactions.
  * **Tight Coupling with Event Broker**: The StateManager's dependence on the EventBroker could limit flexibility if the event-handling mechanism needs to change.
  * **Dependency on Correct Event Subscription**: The StateManager relies on correct event subscription for its operation; missing or incorrect subscriptions could lead to malfunction.

- **StateMachineController**: Manages the state machine's lifecycle, including transitions and event handling.
  - **Motivation:** The StateMachineController class orchestrates the initialization, management, and lifecycle of the application's state machine. This class ensures that the state machine operates smoothly, interacting with language model handlers, input handlers, and output handlers to manage the application's state transitions. It is essential for controlling the flow of the application, responding to events, and ensuring that state transitions are executed correctly.
  - **Requirements:**
    * **Centralized Control**: Should centralize the management of the state machine to ensure that all related operations are handled consistently.
    * **Modularity**: Should abstract the creation and management of the state machine and its components to promote modularity and ease of maintenance.
    * **Event-Driven Architecture**: Should leverage an event-driven architecture, subscribing to and handling relevant state machine events, to allow for responsive and flexible state management.
    * **Scalability**: Should allow for easy scaling by supporting different state machines, language model handlers, and input/output handlers.
    * **Flexibility**: Should support both starting from scratch or resuming from a paused state, providing flexibility in managing the application flow.
  - **Concerns:**
    * **Dependency on Event Flow**: The effectiveness of the StateMachineController is tightly coupled with the correct flow of events; any disruptions or errors in event handling can impact state transitions.
    * **Complexity in Setup**: Setting up the controller requires a deep understanding of the components involved, including the state machine, LLM handlers, and event broker.
    * **Error Propagation**: Errors in one part of the state machine or its components (e.g., LLM handler) can propagate through the controller, potentially causing disruptions in the entire application's flow.

- **ResourceManager**: Manages application resources, ensuring efficient access, updates, and maintenance.
  - **Motivation:** The ResourceManager class is designed to manage the resources within the application's state machine. Resources are critical pieces of data that the state machine uses during its operations. By centralizing the management of these resources, the ResourceManager ensures that the state machine can efficiently access, update, and maintain the necessary data, especially when handling language updates or other dynamic changes in the application state.
  - **Requirements:**
    * **Centralized Resource Management**: Should provide a centralized interface for accessing and updating resources, ensuring consistency and reducing the potential for errors.
    * **Event-Driven Updates**: Should subscribe to relevant events, such as language updates, to dynamically adjust resources and ensure the state machine has the most up-to-date information.
    * **Ease of Access**: Should offer straightforward methods for checking the existence of resources, retrieving them, and updating them, simplifying interaction with the state machine's memory.
    * **Integration with StateMachine**: Should be tightly integrated with the state machine, allowing seamless management of resources in response to state transitions and other state machine activities.
    * **Customizable Resource Handling**: Should allow for dynamic setting of resources to provide a flexible and adaptable approach to managing application data.
  - **Concerns:**
    * **Tight Coupling with StateMachine**: The ResourceManager is tightly coupled with the state machine's memory structure, which could limit its flexibility or make it challenging to reuse in different contexts.
    * **Dependency on Correct Event Flow**: The effectiveness of the ResourceManager relies on the correct and timely flow of events; any disruption could lead to outdated or incorrect resource states.
- **StatePersistenceManager**: Handles saving and loading the application's state, ensuring continuity across sessions.
  - **Motivation:** The StatePersistenceManager class is designed to handle the saving and loading of application state. This functionality is crucial for ensuring continuity in applications that involve complex state management, as it allows the application to resume from a saved state after interruptions or across sessions. The class integrates with various event-driven components, ensuring that state changes are captured and persisted effectively.
  - **Requirements:**
    * **Continuity**: Should allow the application to save its current state and later resume from where it left off, ensuring continuity.

    * **Event-Driven Persistence**: Should subscribe to various events to ensure that changes in state are captured and persisted in real-time.
    * **Modularity**: Should be modular and integrate smoothly with other components via the event broker, promoting a clean separation of concerns.
    * **Language Handling**: Should manage and update language settings dynamically, ensuring that the correct language is maintained across sessions.
    * **Flexibility**: Should support saving and loading state either through execution codes or file paths, providing flexibility in how state persistence is managed.
  - **Concerns:**
    * **Complexity in Error Handling**: The process of saving and loading states, especially during complex state transitions, can be prone to errors that require robust handling mechanisms to avoid content corruption and application interruptions.
    * **Dependency on Correct Event Flow**: The effectiveness of the StatePersistenceManager is highly dependent on the correct flow and handling of events; missing events can lead to incomplete state persistence.

- **DebugHandler**: Provides debugging support by monitoring the state machine's activities.
  - **Motivation:** The DebugHandler module is designed to facilitate the debugging of the application's state machine by binding specific output handlers to key aspects of the state machine's operations. By allowing developers to observe the prompt, resources, and stack during execution, this module enhances the ability to trace and understand the flow of the application, making it easier to identify and resolve issues.
  - **Requirements:**
    * **Enhanced Debugging Capabilities**: Should bind specific output handlers to the state machine's prompt, resources, and stack to provide detailed insights into the state machine's operations, aiding in the debugging process.
    * **Modularity**: Should be modular and easily integrated or removed from the application, depending on the debugging needs.
    * **Flexibility in Output Handling**: Should use output handler interfaces to allow for flexible routing of debugging information, whether it be to a console, GUI, or other logging mechanisms.
    * **Improved Traceability**: Should provide detailed output of the state machine's internal workings to help developers trace the execution flow and pinpoint where issues may arise.
    * **Ease of Integration**: Should be quickly set up by binding it to the state machine without needing extensive changes to the existing application architecture.

- **Concerns:**
  * **Limited Scope**: The DebugHandler is focused on the state machine's prompt, resources, and stack. If other aspects of the application need debugging, additional mechanisms may be required.
  * **Manual Binding Required**: The output handlers need to be manually bound to the DebugHandler, which might be an additional step for developers and could be prone to oversight if not set up correctly.

AppConfig Module

Motivation: The AppConfig module is responsible for loading and managing the configuration settings of the application. It centralizes the application's configuration data, including supported languages, state definitions, and handler mappings, providing a structured and consistent approach to initializing and managing the application's states and behaviors.

Components (Figs. 3.13 and 3.14):

- **AppConfig**: The main class that loads the configuration from a YAML file, initializes application states, and stores various configuration settings such as the application's name, supported languages, and instructions.

Requirements:

- **Centralized Configuration**: Should centralize all application configuration settings to make it easier to manage and modify the application's behavior.
- **Modularity**: Should allow for modular definition of states and behaviors, enabling easy updates and extensions.
- **Flexibility**: Should allow custom handlers to be defined and associated with states, providing a flexible approach to defining application behavior.
- **Consistency**: Should ensure that the application's states and configurations are consistently initialized and managed.
- **Ease of Use**: Should load configurations from a YAML file to simplify the process of defining and updating application settings.

Concerns:

- **Dependency on YAML Structure**: The application relies on the correct structure of the YAML configuration file; errors in the YAML file can lead to initialization failures.
- **Complexity in State Initialization**: Initializing states and their associated handlers can be complex, particularly for large applications with many states.
- **Limited Dynamic Configuration**: Changes to the configuration typically require editing the YAML file and restarting the application, limiting dynamic configuration capabilities.
- **Scalability Concerns**: As the number of states and configurations grows, managing and understanding the configuration file can become cumbersome.

EventBroker Module

Motivation: In applications that involve multiple components interacting with each other, managing events and communication between these components is crucial. The EventBroker module abstracts the event handling mechanism, allowing for decoupled communication between various parts of the application. This ensures that components can respond to events without needing to be directly aware of the other components that triggered them, enhancing modularity and flexibility.

Components (Figs. 3.13 and 3.14):

- **EventType**: An enumeration that defines the different types of events that can occur within the application.
- **EventMessage**: An abstract base class representing the messages or data payloads that accompany events.
- **EventResponse**: An abstract base class representing responses that can be returned by subscribers when an event is processed.
- **EventBroker**: The core component that manages subscriptions and notifies subscribers when events occur. It handles the subscription, unsubscription, and notification processes.

Requirements:

- **Decoupled Communication**: Should promote loose coupling by ensuring components do not need to be aware of each other.
- **Scalability**: Should allow new event types and subscribers to be added without modifying existing components.
- **Modularity**: Should encapsulate the event handling logic within the EventBroker to make the system more modular.
- **Flexibility**: Should allow the system to easily adapt to new requirements by adding or removing subscribers dynamically.

Concerns:

- **Complexity in Debugging**: Tracing the flow of events can be challenging, especially when multiple subscribers are involved.
- **Performance Overhead**: Managing subscriptions and notifying subscribers can introduce performance overhead, particularly with a large number of events and subscribers.
- **Potential for Unhandled Events**: If no subscribers are registered for a particular event, the event might go unhandled, leading to potential issues and hard to find issues.
- **Error Handling**: Errors in subscriber methods can disrupt the notification process, requiring careful error handling mechanisms.

Logging Module

Motivation: Logging is crucial for debugging, monitoring, and maintaining an application. Configurable logging allows for different logging levels and outputs, depending on the environment and needs.

Components (Figs. 3.13 and 3.14):

- **Logging Setup**: Configures logging for the application, including file and console logging.

Requirements:

- **Debugging**: Should provide detailed logs to help identify issues and understand the application's behavior.
- **Monitoring**: Should provide insights into the application's performance and usage through logs.
- **Configurability**: Should allow logging to be configured for different environments.

Concerns:

- **Performance Impact**: Extensive logging can impact performance.
- **Log Management**: Managing log files, especially in long-running applications, can be challenging.

3.3.2 Application Development Facilities

State Update Handlers Module

Motivation: Each state may require specific logic to update its state based on user input and LLM responses. Modular update handlers encapsulate this logic, keeping the state machine clean and manageable. These handlers can be referenced in StateConfig's YAML file to bind during system initialization.

Components:

- **Update Handlers**: Defines various functions for updating states based on user input and LLM responses. These are bound to a state upon configuration.

The State Machine will call the state's update handler with the following parameters and expects the following return values:

```
FUNCTION update_handler(state, resources, llm_handler)
    INPUT:
        state - The current state object
        resources - A list of resources relevant to the current state
        llm_handler - The language model handler responsible for generating outputs
    OUTPUT:
        A tuple containing:
        - A list of ResourceAction objects (optional)
        - A state to dynamically push into the stack (optional)
    BEGIN
        // Application Developer Code
    END
```

This setup allows the update handler to access the state it is bound to, as well as the State Machine's resources and capabilities through llm_handler.

Requirements:

- **Encapsulation**: Should encapsulate state-specific logic in dedicated handlers.

- **Reusability**: Should allow handlers to be reused across different states or applications.
- **Maintainability**: Should decouple state logic from the state machine to simplify maintenance.

Concerns:

- **Complexity**: Handling complex logic within state update handlers can make the code harder to read and maintain.

WebSocket and Flask Bootstrapping Module

Motivation: Real-time interaction with web clients requires WebSocket integration. Flask provides a robust framework for building web applications, and SocketIO adds real-time WebSocket capabilities.

Components:

- **BaseAppNamespace**: Defines the WebSocket namespace for Flask-SocketIO integration.

Requirements:

- **Real-time Communication**: Should enable real-time interaction with web clients.
- **Integration**: Should combine Flask and SocketIO to handle web requests and real-time communication.
- **Customization**: Should allow for custom JSON serialization/deserialization.

Concerns:

- **Scalability Issues**: WebSocket connections can be resource-intensive.
- **Tight Coupling**: Integration with Flask and SocketIO means switching frameworks would require significant changes.
- **Complexity in Real-time Handling**: Handling real-time data and synchronization can be complex.

Tk Bootstrapping Module

Motivation: Creating a consistent and interactive user interface in Tkinter requires reusable components. Abstracting UI components ensures a consistent look and feel across the application.

Components:

- **UIComponents**: Abstract base class for creating the main UI components in Tkinter.
- **VertUIComponents**: Implementation of **UIComponents** for a vertical layout.
- **HorizUIComponents**: Implementation of **UIComponents** for a horizontal layout.

Requirements:

- **Consistency**: Should ensure a consistent user interface across different parts of the application.

- **Reusability**: Should provide common UI components that can be reused, reducing code duplication.
- **Modularity**: Should be modular, making each component easier to manage and extend.

Concerns:

- **Limited Flexibility**: Customizing the UI components beyond the provided base classes may be difficult.
- **Dependency on Tkinter**: Tightly coupled with Tkinter, limiting flexibility to switch to other UI frameworks.
- **Performance**: Tkinter may not be the most performant UI library.

Tk Debug Window Bootstrapping Module

Motivation: Developers need tools to monitor and debug the application state in real-time. A dedicated debug window provides insights into the internal workings of the application.

Components:

- **DebugWindow**: Creates a separate window in Tkinter to display debugging information.

Requirements:

- **Real-time Monitoring**: Should allow developers to monitor the application state in real-time.
- **State Management**: Should provide tools to export and import the application state.
- **Separation of Concerns**: Should ensure that debugging tools are separated from the main application logic.

Concerns:

- **Limited to Tkinter**: The debug window is tightly coupled with Tkinter.

3.4 Future Work

While the CreatAIlity framework serves as a robust proof of concept for integrating Finite State Machines (FSMs) with Large Language Models (LLMs) to foster children's narrative creativity, several areas warrant further research and development to enhance the system's functionality, scalability, and reliability.

3.4.1 Enhanced Testing Strategies

As this work primarily serves as a proof of concept, the focus has been on validating core functionalities. However, to ensure the system's reliability in production environments, more comprehensive testing is required:

Unit Testing

Future work should expand the scope of unit tests to ensure that:

- All individual components and modules are rigorously tested in isolation.
- Potential issues are identified and resolved early, particularly in critical modules like the `StateMachineController` and `ResourceManager`.

Integration Testing

Although basic integration testing has been conducted, future efforts should include:

- Extensive tests covering the full range of interactions between modules, including LLMs, state transitions, and resource management.
- Validation of complex state dependencies and interactions to ensure seamless operation.

Performance Testing

As the framework scales to handle more complex scenarios, future work should focus on:

- Stress testing the system under heavy loads to ensure it can handle multiple simultaneous interactions and large state machines.
- Optimizing performance to minimize latency and overhead in state management and LLM interactions.

3.4.2 Improved Error Handling and Fault Tolerance

The current implementation includes basic error handling mechanisms, but future work should focus on enhancing these aspects:

Error Handling

To ensure system stability, future developments should include:

- Comprehensive error handling strategies that manage unexpected failures gracefully.
- Mechanisms to capture, log, and respond to errors in real-time, maintaining stability under adverse conditions.

Fault Tolerance

To enhance reliability, future iterations of the framework should incorporate:

- Fault tolerance features such as redundancy and failover mechanisms.
- Strategies that allow the system to continue operating smoothly despite component failures.

3.4.3 Scalability and Performance Optimization

The current implementation serves as a proof of concept, but future work should explore:

Scalability Considerations

To manage larger workloads, future work should include:

- Strategies to enhance the scalability of the framework, such as optimizing state management algorithms and exploring distributed state machines.
- Leveraging cloud-based solutions for handling increased complexity and user interactions.

Performance Optimization

For improved performance, future efforts should focus on:

- Refining resource management techniques to reduce overhead.
- Optimizing LLM interaction processes to minimize latency.

3.4.4 Security Enhancements

While overall security is primarily managed by the application leveraging the CreatAIlity framework, it is important to recognize that the framework itself may persist potentially sensitive content and user interactions. In this context, it is crucial that the framework provides robust support for the following security measures:

Data Security

Future work should prioritize:

- Integration of data security measures, such as encryption for data in transit and at rest.
- Ensuring that sensitive user data or educational content is protected.

Access Control

To secure system access, future work should implement:

- Robust access control mechanisms to ensure that only authorized users can interact with specific parts of the system.
- Strategies to manage user permissions and roles effectively.

3.4.5 Proof of Concept Considerations

It is important to acknowledge that the current version of CreatAIlity serves as a proof of concept, aimed at demonstrating the feasibility of integrating FSMs with LLMs for interactive storytelling while leveraging important concepts which aim to improve its efficiency and overall results. Therefore, certain aspects such as comprehensive testing, advanced error handling, fault tolerance, and security were not the primary focus of this study. However, these elements should be considered paramount for any future work that seeks to transition the framework from a conceptual prototype to a fully-fledged educational tool.

In conclusion, while the CreatAIlity framework has demonstrated its core concepts, the areas outlined above represent relevant paths for future exploration. Addressing these aspects will ensure greater reliability, scalability, and security, making CreatAIlity a more robust and versatile tool for fostering narrative creativity in children.

3.5 Conclusion

In this chapter, the architecture and design of the CreatAIlity framework have been detailed, an innovative approach that integrates Finite State Machines (FSMs) with Large Language Models (LLMs) while making prompt chaining, microtasks and *incipits* central to application design to foster children's narrative creativity. The modular design of the framework, combined with a stack-based control mechanism and resource-based state management, provides a structured yet flexible foundation for interactive storytelling. Each component, from the `StateMachineController` to the `ResourceManager`, has been carefully crafted to support the seamless management of narrative flow, resource handling, and state transitions.

The integration of prompt chaining and microtasking techniques further enhances the system's ability to manage LLM outputs, ensuring that the generated narratives are coherent and aligned with educational objectives. By breaking down the storytelling process into smaller, manageable tasks, CreatAIlity facilitates a controlled interaction between the FSM and LLMs, allowing for dynamic and engaging content creation.

While the current implementation serves as a robust proof of concept, the Future Work section has highlighted several areas that require further exploration to transition CreatAIlity from a conceptual prototype to a fully-fledged educational tool. Key areas for improvement include the expansion of unit and integration testing, the enhancement of error handling and fault tolerance mechanisms, and the optimization of scalability and performance. Additionally, future work should prioritize the integration of comprehensive security measures to ensure data protection and access control in real-world educational environments.

Another important consideration in the design process is to minimize dependency on specific technologies, including the State Machine module. While the State Machine is central to the framework, alternative approaches, such as Behavior Trees, can also serve as the core engine. Behavior Trees provide an efficient method for creating complex, modular, and reactive systems, and have been successfully applied in various fields, including video games and robotics as presented by Colledanchise and Ögren, 2018. This flexibility allows for the exploration of different strategies without being constrained by a single technological choice.

Overall, here the groundwork for a versatile and extensible framework that can adapt to various storytelling contexts and educational goals has been laid. The proposed future enhancements would not only improve the system's robustness and reliability but also pave the way for its broader application in educational and potentially other technologies. As the CreatAIlity framework continues to evolve, it holds the potential to significantly impact the way narrative creativity is fostered in children, providing them with a powerful tool for exploring and developing their storytelling skills.

Chapter 4

Implementation and Evaluation

Following the design and modeling of the framework, this section delves into the technical choices and decisions made throughout the development process, alongside the challenges and limitations encountered during implementation, testing, and evaluation. Additionally, a comprehensive overview of the StoryBuilder PoC application is provided, highlighting its functionalities and setting the stage for its subsequent evaluation and the discussion of results.

4.1 Framework Implementation

4.1.1 Technical Choices

The technical choices during the framework's implementation were driven by the need to balance flexibility, performance, and ease of development. Python was selected as the primary programming language due to its extensive library ecosystem, which offers robust support for machine learning, natural language processing, and rapid prototyping. Its dynamic typing and simplicity enabled quick iteration during development, crucial for exploring various design alternatives and integrating advanced features like large language models (LLMs). Despite Python's known limitations in runtime performance and type safety, its advantages in development speed and accessibility made it the ideal choice for this proof of concept.

While Python emerged as the best option, other languages were considered. Java, known for its strong type system and enterprise-level reliability, was ultimately set aside due to its limited integration with LLM libraries. TypeScript, popular in web development for its strong typing, was also considered but deemed less suitable due to its historically less stable ecosystem for the specific objectives of this framework. Ultimately, Python's mature ecosystem and strong community support tipped the balance in its favor.

Python's simplicity and comprehensive ecosystem were key to enabling rapid prototyping, a necessity for this proof of concept. The language's design reduces the need for boilerplate code, significantly speeding up development. While Python's interpreted nature could pose challenges in larger projects, this factor did not impact this project, where the primary performance bottlenecks were API calls and LLM processing times, not Python's execution speed. Therefore, Python's performance was sufficient to meet the project's requirements.

For the user interface, Python's Tkinter library was chosen for its simplicity and ease of integration, despite being somewhat outdated. This choice facilitated the rapid development of a functional prototype, adequate for the proof of concept phase.

For web integration, Flask and SocketIO were selected for their lightweight, flexible nature, enabling seamless communication between the application and web components.

The architecture was designed for versatility, managing dynamic transitions and adapting to different states within the application. However, the decision to prioritize a structured, deterministic process flow introduced constraints, such as linearity and challenges with state update handler reusability. These choices aimed to ensure a predictable, controlled execution environment, essential for maintaining system reliability. The balance between flexibility and structure was carefully considered, with the architecture leaning towards a more deterministic approach to minimize complexity and errors in state transitions.

The selection of Llama3 models for the storytelling tasks was driven by their strong performance, availability, and seamless integration with the Python-based framework. To accommodate different needs, Llama3 8B and Llama3 70B were primarily chosen for testing and evaluation. These models were integrated using LangChain and deployed via tools like Ollama and Hugging Face's TGI server, demonstrating the framework's capacity to support a wide range of connectors with minimal integration effort. Additionally, the framework's flexible interface ensures compatibility with various LLM providers, allowing users to select the models best suited to their specific requirements.

In summary, the technical choices made during the framework's development balanced practicality and performance, leveraging Python's strengths in rapid development and LLM integration. While certain limitations were acknowledged, these choices were appropriate for the project's scope and objectives, providing a solid foundation for the framework's implementation and evaluation.

4.1.2 Challenges and Limitations

Architecture

The proposed architecture is designed to offer significant versatility, even in its ability to manage dynamic transitions and adapt to various states within a process. However, despite these strengths, several challenges and limitations must be acknowledged, which may impact its broader applicability.

One of the key strengths of the architecture is its ability to facilitate deterministic transitions. However, optional states can be seamlessly integrated into the execution sequence by any state, and these states can, in turn, return new states to be added to the process, along with optional resources. This flexibility allows for a more adaptable process flow. However, this dynamic approach can sometimes diverge from the intended use of the architecture, particularly by reducing the reusability of state update handlers. These handlers were initially designed to be highly reusable components, but the introduction of dynamic transitions can complicate this reusability and may limit the system's ability to support truly dynamic state changes. Although the architecture primarily aims to produce a structured, deterministic process flow, the ability to integrate dynamic transitions remains a valuable feature for enhancing system adaptability.

A significant limitation of the architecture is its inherent linearity. The framework is structured to follow a sequential process where each state must be completed before transitioning to the next. Even with the inclusion of dynamic states, the framework does not easily accommodate the skipping of states or transitioning to later states without completing the required preceding ones. This linear progression, while deliberate in design to ensure a controlled and predictable process flow, imposes a hard limit on the framework's use in contexts that require non-linear or conditional transitions. For example, in complex workflows or user-driven applications, the inability to jump between states based on real-time inputs or changing conditions can restrict the flexibility of the system, making it less suitable for certain use cases that demand more fluid and responsive state management.

Another critical challenge is the binding of state update handlers to the states they manage. These handlers are defined separately in the code, which can reduce visibility and make it harder for developers to track and manage them, especially as the project scales. When the expected behavior of a state changes, not only must a new update handler be implemented, but it must also be correctly bound in the configuration file and loaded as a valid handler. This process, while manageable in smaller projects, can become cumbersome in larger teams or projects with multiple developers. The weak binding between the state, its configuration, and the handler implementation increases the risk of errors and necessitates additional oversight. This issue underscores the importance of maintaining thorough documentation and implementing robust testing procedures to ensure consistency and prevent potential problems as the project evolves.

The linearity limitation, in particular, warrants further exploration. While it ensures a structured and deterministic process, it may not be suitable for applications requiring more flexible workflows. To address this, future iterations of the architecture could explore the integration of conditional state transitions or a more advanced state machine that allows for non-linear progression. Such enhancements would enable the architecture to better support complex and dynamic workflows, thereby expanding its applicability across a broader range of use cases.

In summary, while the proposed architecture offers a flexible and structured approach to managing state transitions, it also introduces challenges related to dynamic transitioning, linearity, and the binding of state update handlers. Addressing these limitations through future development would be crucial for expanding the architecture's usability in more complex and diverse applications.

Programming Language and Development Process

The implementation of the framework is relatively straightforward. For this proof of concept, Python was selected due to its extensive library ecosystem, cross-platform compatibility, and robust community support, particularly in both academia and industry. Python's ease of use and rapid prototyping capabilities make it particularly well-suited for research-oriented projects where flexibility and speed are critical. Additionally, Python's integration with modern machine learning and large language model (LLM) libraries is unparalleled, making it an ideal choice for this project, which relies heavily on such technologies.

The flexibility and ease of use that Python offers come with certain challenges. The dynamic typing feature, while convenient for rapid development, presented significant difficulties when scaling the project to a larger and more complex architecture. Commonly, dynamically typed languages are criticized for making it harder to catch type-related errors early in the development process (Tratt, 2009; baeldung, 2021). In this project, type-related errors required extensive debugging and tracing, often surfacing late in the development cycle. However, enforcing a consistent typed signature across functions helped detect errors during debugging and, combined with IntelliSense, helped avoid many potential mistakes. This approach, while not foolproof, significantly mitigated the challenges posed by dynamic typing in regard to this characteristic.

Looking ahead, there is potential for further refining the framework by enforcing stricter type checking in Python. Utilizing tools such as mypy for static type checking could help mitigate some of the challenges experienced with dynamic typing, potentially reducing development time and increasing code stability. Moreover, integrating components written in statically typed languages, or even considering a full migration for performance-critical parts, could expand the framework's user base and applicability.

The overall experience of using Python for this project was smooth and pleasurable. All desired and necessary code patterns were successfully implemented, meeting both the functional and non-functional requirements of the framework. While Python might not be the best choice for extremely processing-intensive applications, it was certainly sufficientd - and indeed, quite effectived - for this project's needs. For more demanding computational tasks, other languages might be more suitable, but Python's ease of use and rapid development capabilities made it a strong candidate and ultimately the right choice for this proof of concept.

Large Language Model

This thesis explores the utilization and evaluation of untrained large language models (LLMs) within the context of storytelling, specifically through the framework developed in this research. The primary objective is to assess the capabilities of existing LLMs in generating creative narratives, rather than focusing on training a model specifically for this purpose. There are two main reasons for this approach. First, training a model from scratch or fine-tuning an existing one is prohibitively expensive, both in terms of computational resources and time, which falls outside the scope of this work. Second, the rapid pace at which general-purpose LLMs are being released and improved by large organizations with substantial resources makes it more practical to leverage these pre-trained models. These models are widely accessible through subscription services or self-deployment, enabling integration into frameworks like CreatAIlity without incurring additional training costs. For users requiring more specific functionality, fine-tuning or further training of models is an option, but not a necessity for the framework's intended use.

Llama3 8B was chosen as it represented the latest freely available advancement in LLMs at the time of development and is lightweight enough to run locally and serve as a test platform. Integration was facilitated using LangChain's framework, which connected to the model via the Ollama tool, allowing for local deployment and usage. Llama3 8B performed adequately for the storytelling tasks, but certain limitations were observed. The model was less prolific in its output and more sensitive to prompt variations, necessitating significant effort to craft prompts that consistently generated the desired results. This was particularly challenging when tasks involved analyzing user input for intent detection, where the model's sensitivity to prompt phrasing often led to inconsistent outputs.

Llama3 70B was selected primarily for a small-scale evaluation conducted with participants, which will be detailed in the subsequent section. This model was chosen for its robustness in generating rich outputs and its reduced sensitivity to prompt variations and was readily available for this work usage. The model was deployed on private infrastructure using connectors for Hugging Face's TGI (Text Generation Inference) server. Due to memory limitations on this infrastructure, the model was quantized to 4 bytes, allowing it to load within the available GPU VRAM. While quantization did impact the model's speed and output quality, reducing the richness of the text generated, it was still sufficient to meet the requirements of the evaluation. The trade-off between resource efficiency and output quality was considered acceptable for the purposes of this study.

One of the primary challenges encountered in using LLMs for this framework was the input token limit. For the larger model, the available infrastructure for this proof of concept supported up to 2,000 input tokens. Initially, it was anticipated that this would be sufficient for the smaller, more focused microtasks planned. However, during the evaluation, it became apparent that users took longer than expected to create their stories at each step. This was interpreted as a good outcome, considering the application was indeed stimulating their creativity and engaging them in the task. But this led to a situation where the token limit, which needed to account for the prompt, context (mainly resources from previous steps), and User-AI dialogue, became a significant constraint.

While several strategies exist to mitigate the token limit issue, such as moving window, partial summarization, and others, none of these solutions are perfect. The resource limitations inherent in the use of LLMs will always present challenges, particularly in creative tasks like storytelling, where the richness and continuity of the narrative are crucial. This limitation was not only a theoretical concern but also manifested during testing. To address this, a partial summarization approach was implemented when the token count approached the limit. This strategy provided users with additional space for creativity and helped prevent interruptions due to errors during the story creation process. While this solution enhanced the user experience and mitigated the issue during testing, it is not a perfect solution. For a final product, a more robust approach, such as increasing infrastructure capacity to allow for a higher token limit or implementing more sophisticated summarization techniques, would be necessary to ensure seamless operation and avoid errors.

It is important to note that the limitations and challenges highlighted here are particularly relevant to the storytelling use case, which is inherently demanding in terms of continuity and content volume. However, the framework developed in this research is versatile and can support various structured processes. In some use cases, such as more focused, less content-heavy tasks, the constraints observed may be less significant or even irrelevant. This adaptability underscores the framework's potential for broader applications beyond storytelling, where the specific requirements and limitations of LLMs may differ.

Beyond the specific limitations discussed, other potential challenges associated with using LLMs in this context include the possibility of model biases affecting the narrative, difficulties in maintaining coherence over long interactions, and the challenges inherent in evaluating creative outputs. These issues, while not the primary focus of this work, are important considerations for future research and development in this area.

Future iterations of this work could explore alternative approaches to further enhance the storytelling capabilities of LLMs. For instance, fine-tuning LLMs on storytelling-specific datasets could improve their ability to generate coherent and contextually appropriate narratives. Additionally, hybrid models that combine LLMs with other AI techniques, such as reinforcement learning or narrative planning algorithms, could offer a way to overcome some of the limitations observed in this study.

In conclusion, the selection and use of Llama3 models in this framework demonstrate the viability of leveraging existing LLMs for creative tasks like storytelling. While certain constraints, such as input token limits and sensitivity to prompting, pose challenges, these can be managed with careful design and implementation strategies. The work conducted in this thesis highlights the potential of LLMs in creative applications, while also underscoring the need for ongoing refinement and adaptation to fully realize their capabilities.

4.1.3 Code Metrics

A thorough code metrics can be found in Appendix B.

4.2 StoryBuilder PoC Application

The primary objective of this proof-of-concept application is to demonstrate and validate the underlying architecture using the CreatAIlity framework, while simultaneously achieving the necessary flexibility required to create a structured and adaptable process for a simple story-building application. This application serves as a practical example to showcase how the framework can be utilized to build interactive, state-driven workflows that guide users through a predefined set of steps.

4.2.1 Application Overview

The application is meticulously designed to enhance user interaction with the workflow by providing essential functionalities such as starting a new execution, restarting an existing one, saving the current state, and resuming the process in subsequent sessions. The user is seamlessly guided through a series of predefined steps, culminating in a final stage where the execution is considered complete. This process ensures that users can engage with the application in a structured and intuitive manner, while also benefiting from the ability to pause and continue their work at their convenience.

4.2.2 Developer Responsibilities

When developing an application using the CreatAIlity framework, the primary responsibility of the developer lies in defining the key components of the application's logic. This includes:

- **State Definitions:** Developers must clearly define the various states that the application can occupy. Each state represents a specific point in the workflow, with its own set of rules and conditions that dictate the user's interaction and progression.
- **Resource Definition/Management:** Proper definition and management of resources is essential for the seamless functioning of the application. Resources

may include data, control booleans or any other memory-representable asset required and produced by the application states during their execution.

- ***Incipits*/Prompts:** These are the initial triggers or prompts that bootstrap the state, providing the user with the necessary context or actions to engage with the application. Effective prompts are crucial for guiding the user through the workflow in an intuitive manner.
- **State Completion Mechanism:** A key aspect of the framework is the state completion mechanism, which determines when a state is finished and the application can transition to the next state. Developers leverage the framework's infrastructure to implement these completion mechanics, ensuring that each state concludes only when its specific criteria are met.
- **Update Handlers:** Developers are also responsible for implementing and binding update handlers for each state. These handlers are essential for managing state changes, responding to user inputs, and ensuring that the application behaves as expected during its execution. It is the main processing point for the state logic and the state completion mechanism execution.

User Interface Development

Beyond the core logic, the other significant portion of the development effort is dedicated to creating the user interface (UI). The UI serves as the primary medium through which users interact with the application, whether it be graphical, text-based, or another form of interface. The developer's tasks in this area include:

- **Designing Interaction Elements:** Developers must design and implement the interaction elements that users will engage with. These elements could range from buttons, text inputs and outputs, dropdowns and menus to more complex actions such as exporting the final story, depending on the application's requirements.
- **Interface Stack Binding:** The interaction elements must be properly bound to the interface stack, ensuring that user actions are correctly interpreted and processed by the application.
- **Utilizing Framework Controls:** The developer must make effective use of the framework's control functions or event publishing mechanisms to manage the application's execution flow and feature set. This includes responding to user events, triggering state transitions, and invoking specific functionalities within the framework.

4.2.3 State/Resource and Behavior Design

By modeling the states and expected behaviors, and leveraging the application's capabilities and underlying engine, three front-ends were developed for different environments: Console, Desktop (using Tk), and Web. Through the application facilities provided by the framework, little effort apart from tailoring the interfaces to the specific rendering technology were needed. All core code and configuration was reused and integrated through specialized input/output handlers seamlessly.

The workflow, based on the CreatAIlity framework, encompasses the following states and resources for this specific use case as can be seen in figures 4.1 and 4.2. This sequence ensures that each state builds on the resources provided by the previous states, maintaining a coherent and logical narrative structure.

The configuration and setup of State and Resources are done by the config.yaml file and an update handlers python script containing methods as defined in section 3.3.2.

The yaml file follows this structure:

```
app:
  name: "Application Name"
  supported_languages:
    - "Language Code 1"
    - "Language Code 2"
  initial_state: "initial_state_name"
  instructions:
    language_code_1: "Instruction text in language 1"
    language_code_2: "Instruction text in language 2"
  states:
    state_name:
      requires:
        - "required_resource_1"
        - "required_resource_2"
      provides: "provided_resource"
      provides_content_holder:
        language_code_1: "Content holder label in language 1"
        language_code_2: "Content holder label in language 2"
      update_handler_name: "handler_function_name"
      user_prompt:
        language_code_1: "User prompt in language 1"
        language_code_2: "User prompt in language 2"
      prompt:
        language_code_1: "Prompt text in language 1"
        language_code_2: "Prompt text in language 2"
      clear_after_input: true/false
      pre_fill_resource: resource_name/null
      resource_value: value/null
  additional_resources:
    resource_name:
      language_code_1: "Resource text in language 1"
      language_code_2: "Resource text in language 2"
```

- **app:**
  - **name:** The name of the application. This is a string value that typically represents the title or identifier of the application.
  - **supported_languages:** A list of language codes (e.g., "en_US", "es_ES") that the application supports. Each language code corresponds to the localization of instructions, prompts, and other textual content within the application.

  - **initial_state:** The initial state of the application when it starts. This is typically the first state that the application enters when executed.
  - **instructions:** A set of instructions for users, localized by language. The keys are language codes, and the values are the instructional text provided to the user in each respective language.

- **states:**
  - Each item under *states* represents a distinct state in the application workflow. States define the flow of the application, specifying what is required, what is provided, and how the application interacts with the user during that state.
  - **state_name:** The unique identifier for a state in the workflow. This name is used to reference and manage the state within the application.
  - **requires:** A list of resources or conditions that must be satisfied before the state can be entered. These are typically other states or inputs required to proceed.
  - **provides:** The resource or outcome provided upon completing the state. This could be a piece of data or an updated status that is passed to subsequent states.
  - **provides_content_holder:** A mapping of language codes to labels or descriptions for the content provided by this state. This helps in localizing what is provided by the state in different languages.
  - **update_handler_name:** The name of the handler function that manages updates or changes within this state. This function controls the logic of how the state transitions or processes user input.
  - **user_prompt:** Localized prompts presented to the user to guide interaction within the state. The keys are language codes, and the values are the prompts in each respective language.
  - **prompt:** A detailed text that defines what the application prompts the user to do, localized by language. This often includes placeholders for dynamic content like user inputs.
  - **clear_after_input:** A boolean value indicating whether the input should be cleared after the user provides it. If true, the input is cleared, preparing the state for new input.
  - **pre_fill_resource:** Specifies a resource to be pre-filled in this state, if applicable. This could be null if no pre-filling is needed.
  - **resource_value:** The value associated with the resource after the state is completed. This could be dynamic content or null if not applicable.

- **additional_resources:**
  - This section defines additional resources that are used across the application, such as prompts or instructions not directly tied to a specific state.
  - **resource_name:** The identifier for the additional resource. This name is used to reference the resource within the application.

- **language_code_1, language_code_2, etc.:** Localized versions of the resource text. The keys are language codes, and the values are the corresponding text in those languages.

4.2.4 Interfaces and Interaction

Console

The console application represents the most basic implementation of the framework, providing a streamlined set of functionalities due to the inherent constraints of console-based interaction. While it is certainly possible to extend these capabilities and overcome the limitations through innovative approaches, a straightforward input/output interaction was chosen to effectively demonstrate the framework's adaptability within this environment.

Despite its simplicity, the console application delivers a robust experience by facilitating real-time interaction through output streaming of generated text. This design ensures that users receive immediate feedback, enhancing the overall usability of the application. Although the console interface may offer a more limited range of features compared to more sophisticated interfaces, it still provides an enjoyable and efficient experience in scenarios where simplicity and directness are required.

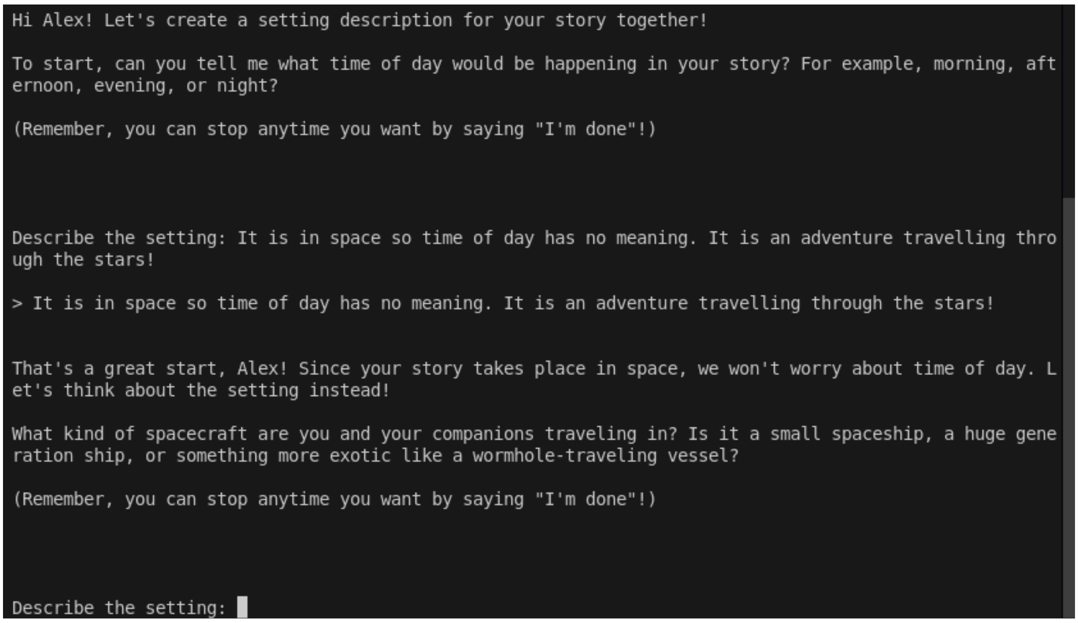

FIGURE 4.3: StoryBuilder Console Application

Desktop

For this version of the application, Tkinter was selected as the desktop GUI renderer, primarily due to its native integration with Python. However, the framework is designed with flexibility in mind, allowing other graphical frameworks to be easily integrated by extending and implementing the necessary logic within the framework's interfaces.

This GUI version offers enhanced flexibility and a broader range of features, including debugging, exporting, and restarting, thanks to its significantly higher interactivity compared to console-based applications. These features were incorporated to

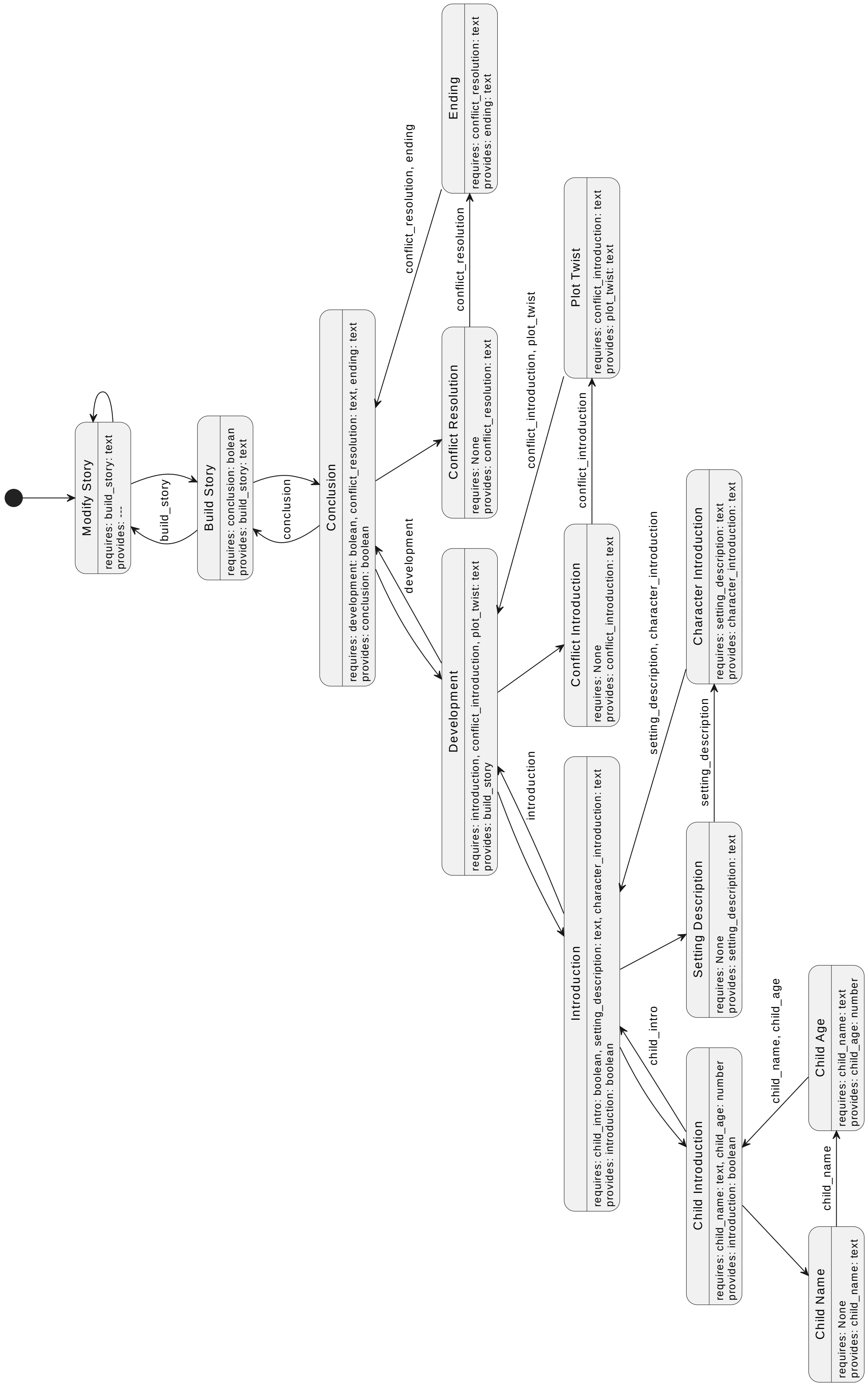

FIGURE 4.1: State diagram for Storybuilder PoC

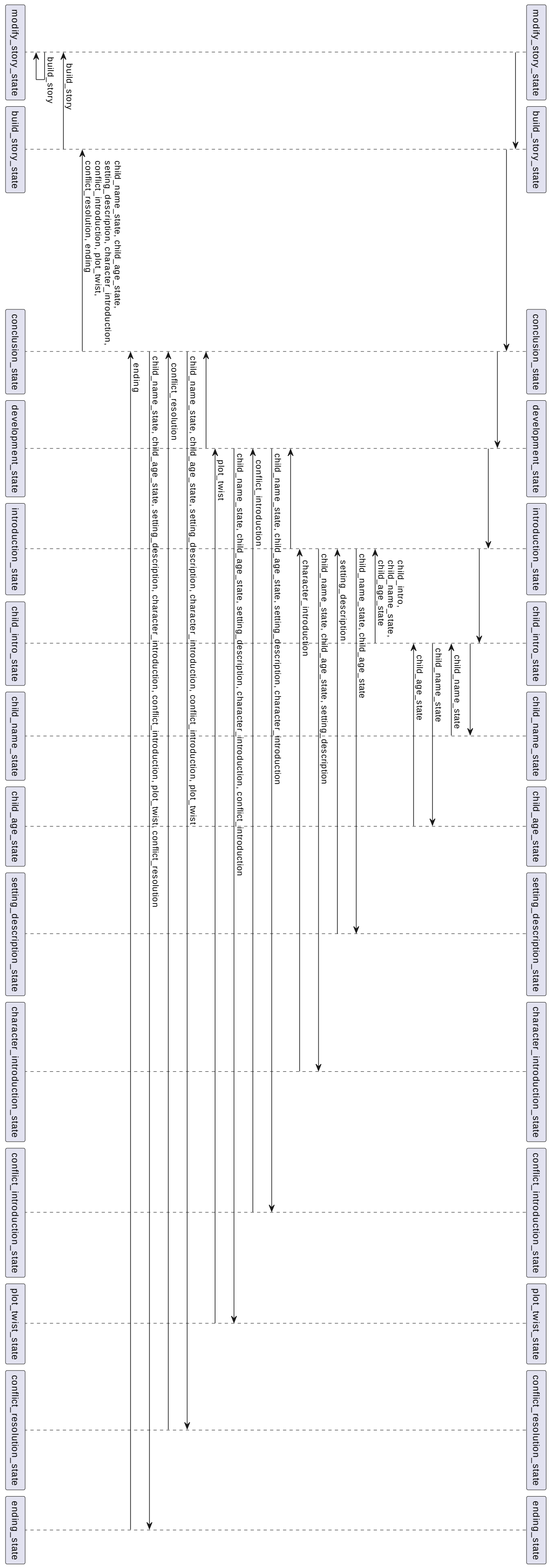

FIGURE 4.2: Resource flow diagram for Storybuilder PoC

demonstrate the framework's adaptability and extensibility, showcasing its capability to function effectively in local environments with native support. This implementation highlights how the framework can be expanded to leverage the full potential of desktop applications, providing a rich and interactive user experience.

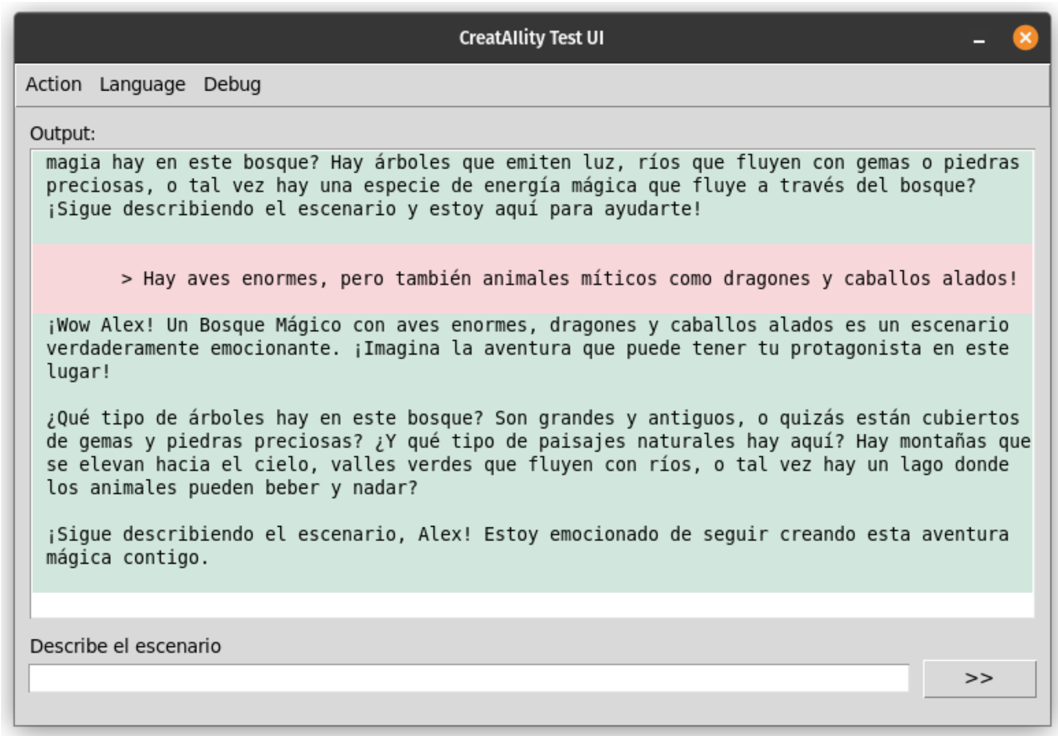

FIGURE 4.4: StoryBuilder Desktop (Tk) Application

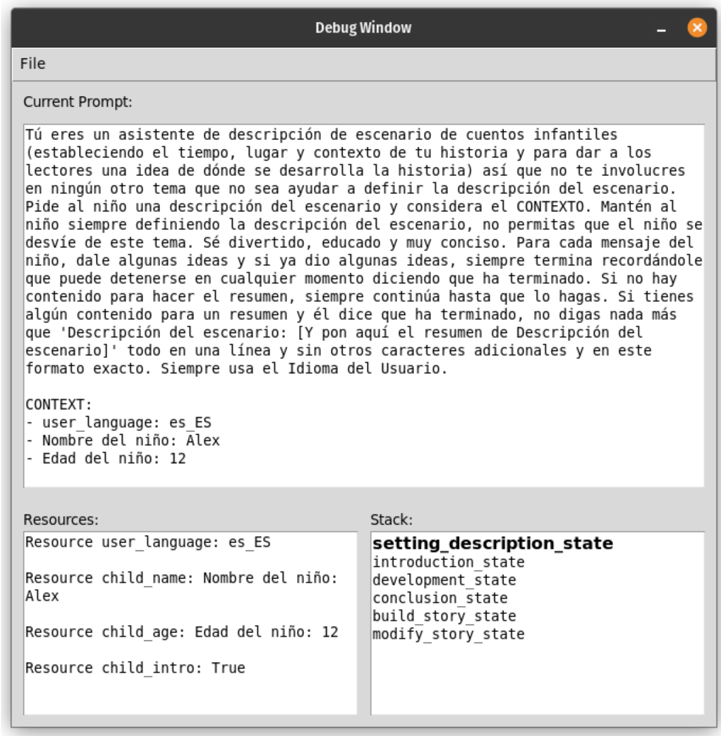

FIGURE 4.5: StoryBuilder Desktop (Tk) Application Debug Window

Web

The web is the dominant platform in today's technological landscape, making the web-based version of the application the most feature-rich and the preferred choice for evaluation. Its inherent advantage of being easily distributed and accessible from anywhere in the world made it ideal for this purpose. Considerable effort was invested in ensuring the interface is visually appealing and intuitive, allowing it to effectively showcase the framework's potential during user testing.

In this implementation, simplicity was key. The objective was not to create a state-of-the-art application with advanced frontend frameworks or an extensive feature set. Instead, the focus was on developing a straightforward, yet highly effective and functional web application. The goal was to deliver a clean and user-friendly interface that meets the needs of the evaluation while remaining accessible and easy to use. This approach ensures that the core strengths of the framework are highlighted without unnecessary complexity.

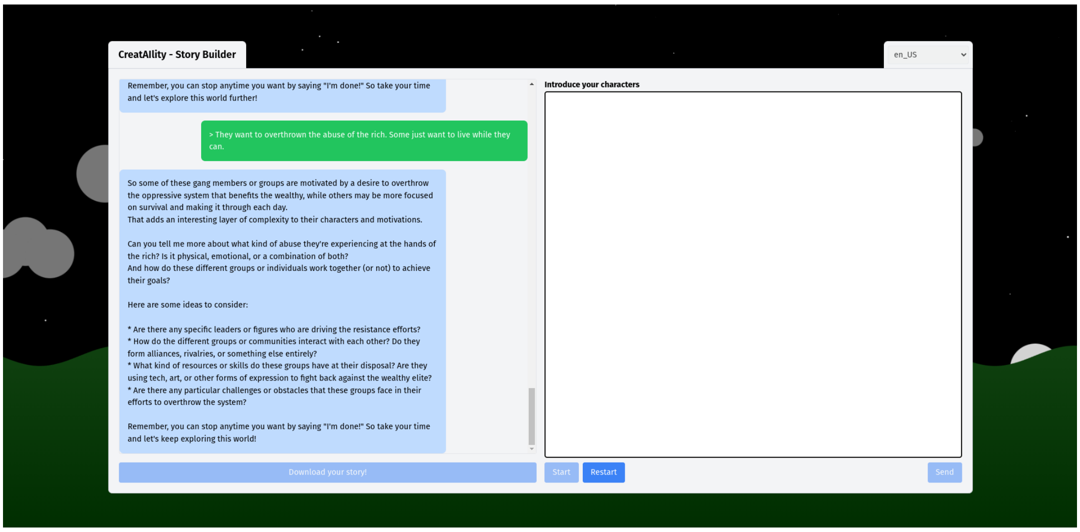

FIGURE 4.6: StoryBuilder Web Application

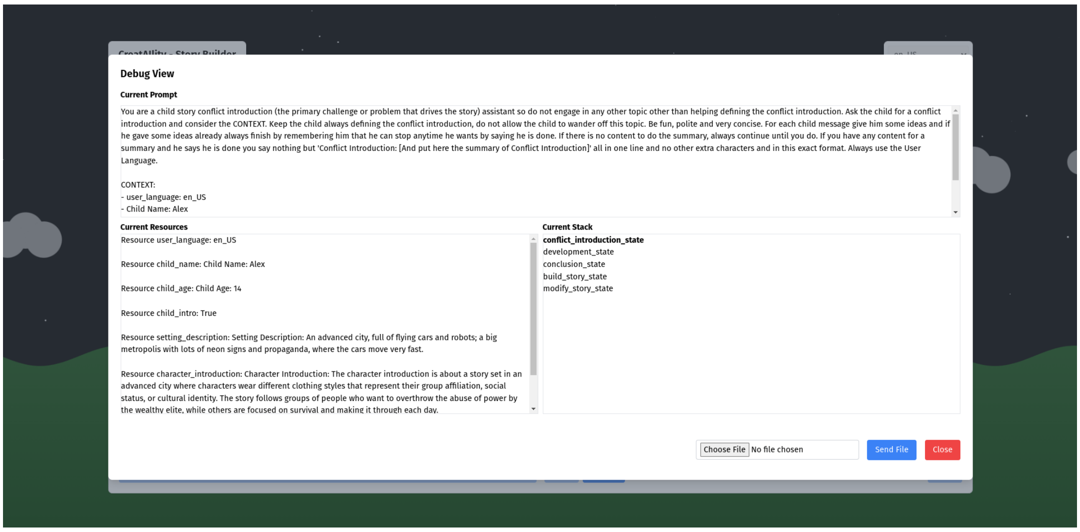

FIGURE 4.7: StoryBuilder Web Application Debug Modal

4.3 Evaluation

4.3.1 Methodology

To evaluate the effectiveness and user experience of the StoryBuilder Proof of Concept (PoC) application, a targeted small-scale experiment was conducted with eight adult participants ranging in age from 30 to 70 years. The decision to focus exclusively on adults was carefully considered, as it allowed the study to circumvent the significant ethical and logistical complexities involved in experimenting with children. Conducting research with minors typically demands extensive ethical reviews, parental consent, and specialized protocols, all of which would have introduced substantial delays and resource demands beyond the scope of this exploratory study.

By selecting adult participants, the research team was able to streamline the recruitment process and proceed with fewer regulatory constraints, facilitating a more efficient and manageable study. The adult demographic was also considered appropriate for this initial phase of testing, as it provided a broad range of perspectives while still maintaining a manageable scope. This approach was deemed adequate for conducting a preliminary evaluation of the application's overall quality, performance, and user experience. The insights gained from this initial study are expected to provide valuable feedback on the application's usability and effectiveness, serving as a foundation for further development and refinement.

Furthermore, the findings from this evaluation will guide subsequent iterations of the application, ensuring that future versions better meet the needs and expectations of a diverse user base. This iterative process of evaluation and refinement is crucial for the eventual goal of optimizing the application for broader deployment.

4.3.2 Evaluation Tools

To ensure a thorough and multi-dimensional assessment of the StoryBuilder Proof of Concept (PoC) application, a carefully selected set of evaluation tools was employed. These tools were chosen based on their ability to capture different aspects of user experience, usability, and workload, providing a comprehensive overview of how the application performs from the perspective of its users. The evaluation tools included the System Usability Scale (SUS), the Game Experience Questionnaire (GEQ), and the NASA Task Load Index (NASA-TLX). Each of these instruments is well-established in their respective domains, offering reliable and valid measures that contribute valuable insights into the application's effectiveness and overall user experience.

The System Usability Scale (SUS) (Brooke, 1996; Lewis, 2018) was utilized to gauge the ease of use and general usability of the application, providing a quick yet robust metric that highlights potential usability issues. Meanwhile, the Game Experience Questionnaire (GEQ) (Poels, Kort, and IJsselsteijn, 2007) was employed to delve into the more subjective and experiential aspects of user interaction, particularly focusing on engagement, immersion, and emotional responses during the use of the application. Finally, the NASA Task Load Index (NASA-TLX) (Hart, 2006; Hart and Staveland, 1988) was included to assess the perceived workload associated with using the application, offering insights into the cognitive and physical demands placed on users.

Together, these tools offer a well-rounded evaluation framework, enabling a nuanced understanding of how the StoryBuilder PoC application performs across different dimensions of user experience. The following subsections detail the specifics of each tool, including their purpose, methodology, and the insights they provide.

System Usability Scale (SUS)

The System Usability Scale (SUS) is a simple, yet effective tool for measuring the usability of a system. Created by John Brooke in 1986, SUS is a ten-item questionnaire that users fill out after interacting with a system, whether it be software, hardware, or a website. Each item is scored on a Likert scale from 1 (strongly disagree) to 5 (strongly agree), with the overall score ranging from 0 to 100 after a specific calculation process.

SUS is valued for its reliability and versatility across various industries and types of systems. It provides a quick assessment of a system's usability by capturing the users' perceived ease of use, satisfaction, and efficiency. The resulting score offers a clear indication of the system's usability, helping developers identify areas for improvement. Although SUS provides a high-level view, it is often used in conjunction with other methods to gain a more detailed understanding of usability issues.

The SUS statements, answered by the use of a likert scale as shown in B.1 in tables B.1 and B.2.

Game Experience Questionnaire (GEQ)

The Game Experience Questionnaire (GEQ) is a comprehensive instrument designed to measure various aspects of a player's experience in digital games. The GEQ is structured into multiple components, each focusing on different facets of the gaming experience, including immersion, flow, competence, tension, challenge, and positive and negative affect. The core module of GEQ captures the general experience during gameplay, while additional modules, such as the Social Presence Module and Post-Game Module, assess specific dimensions like social interaction and the after-effects of gameplay.

GEQ is widely used in game research and development to assess how players interact with and perceive a game. By analyzing responses, developers and researchers can gain valuable insights into how different game elements contribute to the overall experience. This information can then be used to fine-tune game design, ensuring that the final product meets the desired engagement and satisfaction levels for the target audience.

In this evaluation, the core module of GEQ was used, which consists in the following statements and answered in a likert scale as shown in B.2 in tables B.3 and B.4.

NASA Task Load Index (NASA-TLX)

The NASA Task Load Index (NASA-TLX) is a widely recognized tool used to assess perceived workload in various tasks. Developed by NASA, this multi-dimensional rating procedure evaluates a user's subjective experience of workload across six dimensions: Mental Demand, Physical Demand, Temporal Demand, Performance, Effort, and Frustration. Each dimension is rated on a scale from low to high, and the overall workload score is calculated by combining these ratings.

The purpose of NASA-TLX is to capture the cognitive and physical demands placed on users during task execution, offering insights into the overall burden experienced. The tool is especially valuable in complex systems where understanding the workload is crucial for optimizing performance and minimizing errors. By quantifying these aspects, NASA-TLX helps in identifying areas where interventions can be made to improve user experience and system efficiency.

The NASA-TLX questions, answered by the use of an adapted likert scale as shown in B.3 in tables B.5 and B.6.

4.4 Results and Discussion

4.4.1 System Usability Scale (SUS)

The System Usability Scale (SUS) results provided offer insights into the usability of the system based on user feedback. The SUS scores are calculated based on a series of questions answered on a 5-point Likert scale, with responses ranging from "Strongly Disagree" (1) to "Strongly Agree" (5). Below is a detailed analysis of the results:

SUS Score Summary

A graphical representation can be found in Fig. 4.8.

Metric	Value
Mean SUS Score	91.25
Standard Deviation	8.45

TABLE 4.1: Summary of SUS Score and Standard Deviation

Interpretation of SUS Items

A breakdown of the distribution of answers can be seen in Fig. 4.9.

1. **"I think that I would like to use this system frequently.":** The majority of users agreed or strongly agreed that they would like to use the system frequently, which is a strong indicator of user satisfaction and perceived usefulness. The presence of neutral responses suggests that while most users are enthusiastic, a couple of users may not see frequent use as essential.

2. **"I found the system unnecessarily complex.":** All users disagreed with the statement, indicating that the system is generally perceived as straightforward and easy to use. This aligns with the high overall SUS score and suggests that complexity is not a barrier to usability.

3. **"I thought the system was easy to use.":** All users found the system easy to use, with the majority strongly agreeing. This further supports the high usability reflected in the overall SUS score, showing that the system's interface and interactions are intuitive.

4. **"I think that I would need the support of a technical person to be able to use this system.":** The strong disagreement with this statement indicates that users felt confident in their ability to use the system without needing additional technical support. This suggests that the system is user-friendly and accessible, even for those who may not be technically inclined.

5. **"I found the various functions in this system were well integrated.":** Users generally found that the system's functions were well integrated, which contributes to its overall usability. A well-integrated system ensures a seamless user experience and efficient task completion.

6. **"I thought there was too much inconsistency in this system.":** The strong disagreement with this statement suggests that users found the system to be consistent in its design and operation. Consistency is a key factor in usability, as it helps users predict how the system will behave.

7. **"I would imagine that most people would learn to use this system very quickly.":** Nearly all users strongly agreed that others would quickly learn to use the system, indicating that the system has a low learning curve. This is crucial for ensuring that new users can become proficient without extensive training.

8. **"I found the system very cumbersome to use.":** All users disagreed with this statement, indicating that they did not find the system cumbersome. This further emphasizes the ease of use and efficiency of the system's design.

9. **"I felt very confident using the system.":** All users felt confident using the system, with an even split between agreement and strong agreement. Confidence in using the system is a crucial indicator of good usability, as it reflects the users' comfort and trust in the system's capabilities.

10. **"I needed to learn a lot of things before I could get going with this system.":** The strong disagreement here indicates that the system is perceived as easy to start using without requiring significant learning or prior knowledge. This suggests that the system is well-designed for quick onboarding and accessibility.

Overall Interpretation

- **High Usability:** The SUS score of 91.25, with a low standard deviation, indicates that users overwhelmingly find the system highly usable. This is further corroborated by the individual item responses, where users consistently rated the system positively.
- **Positive User Experience:** The majority of users found the system easy to use, well-integrated, and not cumbersome or complex. This suggests that the system is well-designed with user experience in mind.
- **Consistency and Integration:** Users found the system to be consistent and its functions well integrated, which are important factors contributing to a seamless user experience.

Final Notes

The simple yet effective approach to the user interface and usability has proven to be successful. Although the application's straightforward nature lends itself to this type of design, it has been well received by the evaluation subjects, indicating its suitability for a younger audience. These results indicate that the system is well-received by users and provides a strong foundation for further development and refinement for more targeted groups.

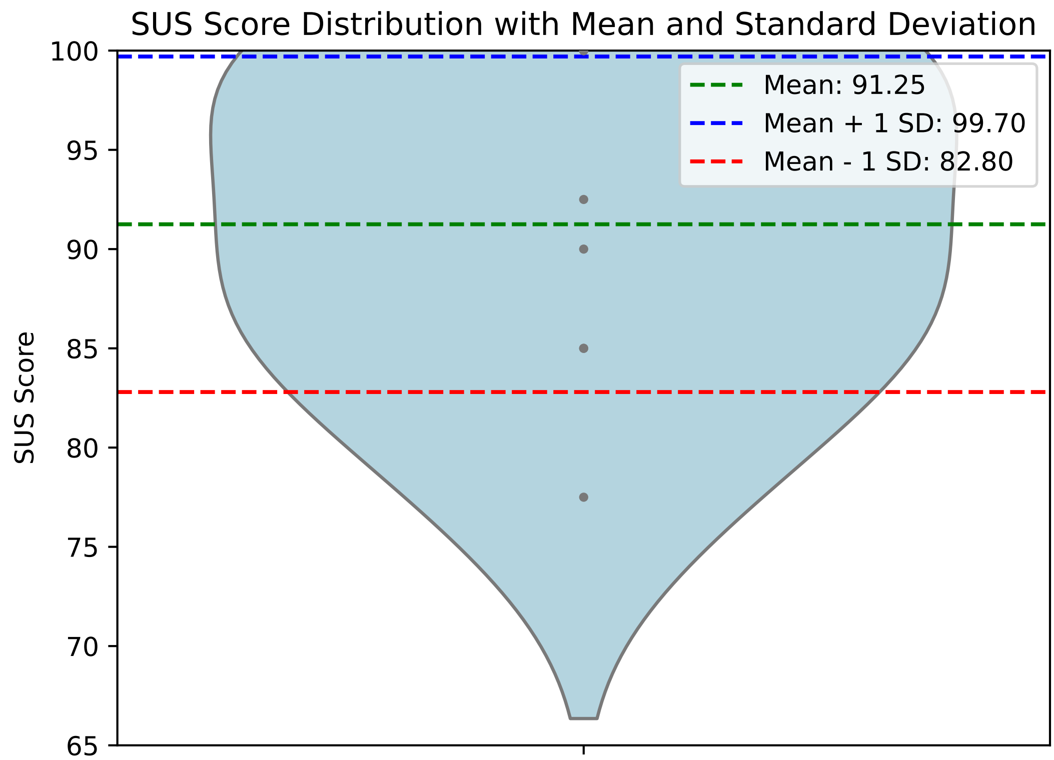

FIGURE 4.8: SUS Score Distribution

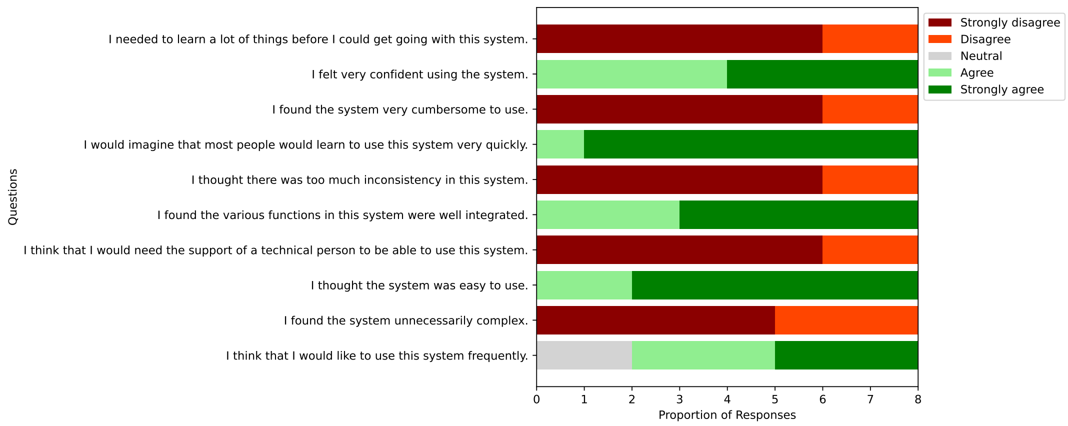

FIGURE 4.9: SUS Response Proportion

4.4.2 Game Experience Questionnaire (GEQ)

Based on the provided Game Experience Questionnaire (GEQ) data and the associated categories from the core module, the results can be analyzed in relation to the specific components of the core GEQ. The core module assesses game experience across seven components: **Competence**, **Sensory and Imaginative Immersion**, **Flow**, **Tension/Annoyance**, **Challenge**, **Negative Affect**, and **Positive Affect**.

GEQ Dimensions Score Summary

An overlay of dimensions for all users can be found in Fig. 4.10, while the mean and standard deviations are shown in 4.11.

GEQ Dimension	Mean Scores	Standard Deviation
Competence	3.1	0.7
Sensory and Imaginative Immersion	3.3	0.7
Flow	2.8	0.7
Tension/Annoyance	0.1	0.4
Challenge	0.7	0.6
Negative Affect	0.3	0.2
Positive Affect	3.7	0.3

TABLE 4.2: Associated Question Items and GEQ Dimensions Scores

Interpretation of GEQ Dimensions

1. **Competence:** The competence scores indicate a moderate sense of skill and effectiveness while using the application. Users generally felt capable, with the mean user scores varying from 2.2 to 4.0. Higher scores reflect a strong sense of mastery, while the lower scores suggest that some users may have struggled with certain aspects of the interaction.

2. **Sensory and Imaginative Immersion:** Immersion scores reveal that most users experienced a moderate to high level of engagement with the application's sensory and imaginative aspects. Scores closer to 4.0 suggest that the application was effective in drawing users into its world, making them feel absorbed in the experience. However, the lower scores indicate that some users were less engaged, possibly due to less captivating content or design elements that did not fully resonate with them.

3. **Flow:** The flow scores suggest that users' experiences of being fully absorbed in the application varied. Some users reported strong flow (score of 4.0), indicating that they were deeply concentrated and lost track of time, which is ideal for creating an engaging experience. Conversely, lower scores (around 2.0) suggest that not all users reached this state, possibly due to interruptions in application usage or mechanics that hindered seamless engagement.

4. **Tension/Annoyance:** The very low tension/annoyance scores (mostly 0.0) suggest that the application was not frustrating for most users. A score of 1.0 indicates minimal tension or annoyance, reflecting a generally positive experience without significant moments of irritation or stress. This suggests that the application is well-balanced in terms of difficulty and did not create negative emotional responses associated with frustration.

5. **Challenge:** The challenge scores are relatively low, with a maximum of 1.4. This indicates that users found the application to be only mildly challenging, if at all. Although the application was designed for small children (8-12 years old), it may also suggest that the application lacks depth or difficulty to fully engage more experienced users. The low challenge might be a factor in the mixed flow scores, as less challenging applications can fail to create the optimal level of difficulty required to sustain flow.

6. **Negative Affect:** The low negative affect scores indicate that users did not experience significant negative emotions, such as boredom or irritation, while playing the application. A score of 0.5 is minimal, reflecting only slight negative feelings, which could be due to brief moments of disengagement or dissatisfaction but were not pervasive throughout the application usage experience.

7. **Positive Affect:** Positive affect scores are consistently high, ranging from 3.2 to 4.0. This suggests that users generally enjoyed the application and felt positive emotions such as happiness, contentment, and satisfaction. High positive affect is a strong indicator of the application's success in creating an enjoyable experience.

Overall Interpretation

The GEQ core module results provide a comprehensive overview of how users experienced the application across key dimensions:

- **Moderate Competence:** Users generally felt moderately competent, indicating that the application offered an appropriate level of challenge for most, though some users might benefit from further difficulty adjustments.
- **Low Challenge:** The relatively low challenge scores suggest that the application may be too easy for some users, potentially limiting its ability to sustain long-term engagement for more experienced people. One must take into account this PoC was designed for small children so a low challenge is to be expected to be reported by adults.
- **Varied Immersion and Flow:** While many users felt immersed and experienced flow, there was notable variability. This suggests that the application successfully engaged some users deeply but might need improvements to ensure a more consistent experience across all users.
- **Low Tension and Negative Affect:** The application effectively avoided causing negative emotions or frustration, which is a positive outcome. This contributes to the overall enjoyment and positive affect reported by users.
- **High Positive Affect:** The consistently high positive affect scores confirm that the application was enjoyable and left users with a generally positive experience.

Final Notes

Conducting an appropriate evaluation with the targeted age group is essential to accurately assess the GEQ, particularly in terms of competence, challenge and flow, which should be carefully tailored to the intended audience. Future research could explore enhancing immersion by incorporating additional elements such as graphical or audio resources, thereby providing a more comprehensive sensory experience.

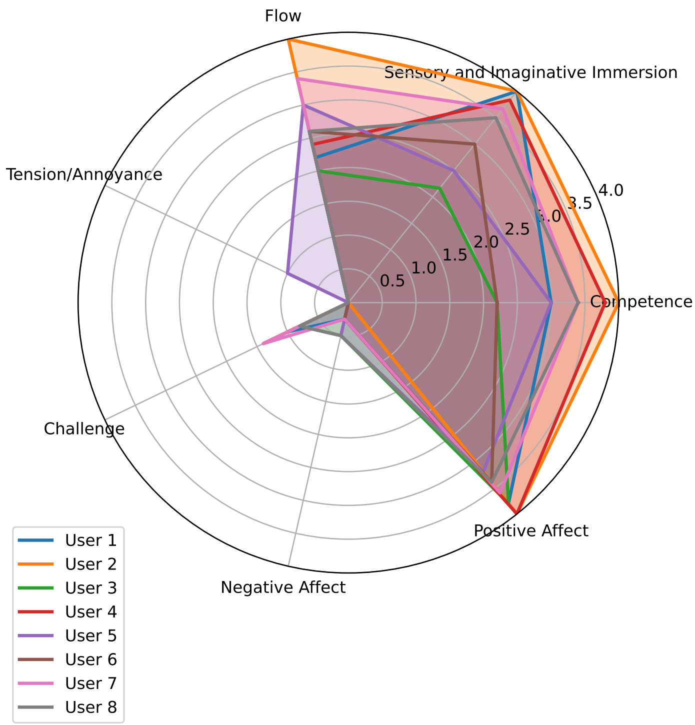

FIGURE 4.10: GEQ Score Radar

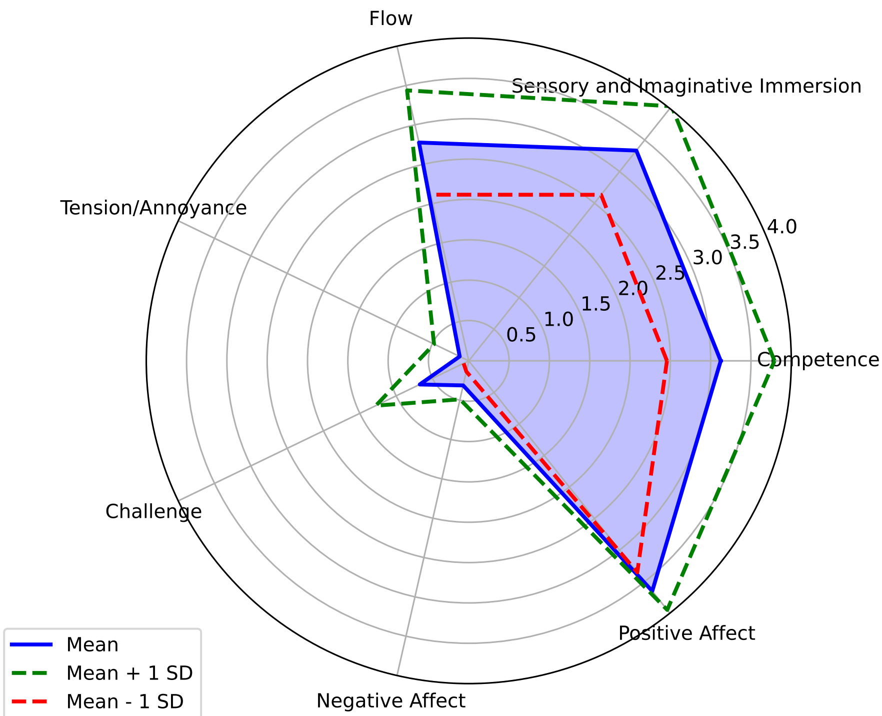

FIGURE 4.11: GEQ Mean-Deviation Radar

4.4.3 NASA Task Load Index (NASA-TLX)

The NASA Task Load Index (NASA-TLX) results provide a multidimensional assessment of perceived workload, normalized to a 0-100 scale. The data includes six dimensions: Mental Demand, Physical Demand, Temporal Demand, Performance, Effort, and Frustration.

NASA-TLX Score Summary

A graphical representation can be found in Fig. 4.12.

Category	Mean	Standard Deviation
Mental Demand	50.0	26.7
Physical Demand	9.4	18.6
Temporal Demand	21.9	16.0
Performance	84.4	12.9
Effort	21.9	16.0
Frustration	0.0	0.0

TABLE 4.3: Breakdown of Mean Scores and Standard Deviations for NASA-TLX Dimensions

Interpretation of NASA-TLX Dimensions

1. **Mental Demand:** The moderate mean score of 50.0 suggests that the tasks required a considerable amount of mental effort. The relatively high standard deviation (26.7) indicates variability in how different participants perceived the mental demand, with some finding it more demanding than others.

2. **Physical Demand:** The low mean score of 9.4 indicates that the tasks were perceived as requiring minimal physical effort. The standard deviation of 18.6, while somewhat high relative to the mean, suggests that most participants found the physical demand to be low, though a few may have perceived it slightly differently.

3. **Temporal Demand:** With a mean score of 21.9, the tasks were generally not perceived as highly rushed or time-pressured. The standard deviation of 16.0 indicates some variation among participants, with most perceiving the temporal demand as low to moderate.

4. **Performance:** The high mean score of 84.4 reflects that participants felt they were highly successful in accomplishing the tasks. The relatively low standard deviation of 12.9 suggests that most participants had a consistent perception of their performance, with generally high satisfaction.

5. **Effort:** The low mean score of 21.9 indicates that participants generally did not have to exert significant effort to achieve their performance levels. The standard deviation of 16.0 shows that this perception was fairly consistent across participants.

6. **Frustration:** A mean score of 0.0, with no variation (standard deviation of 0.0), suggests that the tasks did not induce any feelings of frustration, insecurity, or stress among participants. This is a strong indicator of a positive user experience.

Overall Interpretation

The NASA-TLX results indicate that the tasks were generally well-received by participants, with low levels of perceived workload across most dimensions. Mental demand was moderate, suggesting that the tasks required some cognitive effort, and physical and temporal demands were low as to be expected. Performance was rated highly, indicating that participants felt successful in their tasks, and frustration was non-existent, which is a very positive outcome.

Final Notes

The low physical and temporal demands indicate that tasks were appropriately designed to minimize physical strain and time pressure, making them suitable for a younger demographic. However, while the mental demand was moderate as to be expected, further adjustments to streamline task complexity could reduce cognitive load, enhancing overall performance and satisfaction depending on the target group. The high performance scores and absence of frustration highlight the effectiveness of the current task design. Future iterations should focus on maintaining these strengths to ensure continued user satisfaction and to accommodate the unique needs of children, who may be more sensitive to excessive cognitive or physical demands.

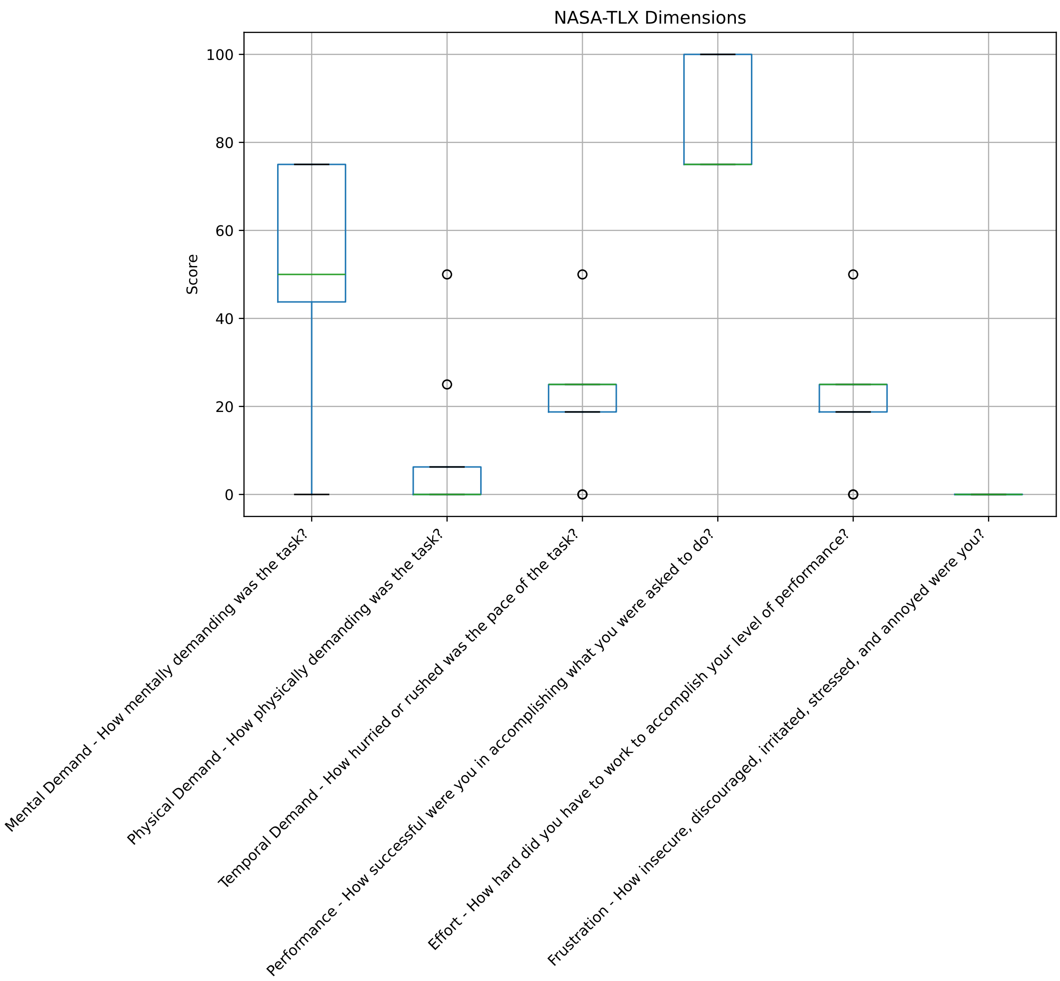

FIGURE 4.12: NASA-TLX Score Dimensions

Chapter 5

Conclusion

The development of the CreatAIlity framework marks a significant advancement in the integration of artificial intelligence with educational technology, specifically designed to foster creativity in children through guided storytelling. By leveraging key concepts and combining the generative power of Large Language Models (LLMs) with the structured control of Finite State Machines (FSMs), CreatAIlity effectively balances the spontaneity of AI-driven content generation with the educational need for coherence and structure in narrative construction. Its modular and flexible design not only allows for experimentation with alternative flow control technologies, such as Behavior Trees or other State Machine approaches, but also facilitates integration with diverse tech stacks for interacting with various language models. Additionally, the framework is designed to be easily extendable, enabling straightforward integration with other media producers, such as audio or image generators, further enhancing its versatility and potential in educational applications.

5.1 Summary of Achievements

This work has successfully achieved several key objectives. Primarily, it has demonstrated the feasibility and effectiveness of integrating LLMs with FSMs to create a tool that is both creatively engaging and pedagogically sound. While targeted evaluation is necessary to fully validate the pedagogical outcomes, the framework has shown that various approaches can be tailored to meet different educational goals, as evidenced by the validation of this use case. The framework empowers children to engage in narrative creation within a scaffolded environment, allowing them to explore their imagination while receiving guidance to ensure their stories are coherent and meaningful. The use of *incipits* and prompt chaining has proven particularly effective in maintaining narrative flow, offering a balanced combination of flexibility and control in the storytelling process.

Moreover, the framework's design emphasizes modularity and extensibility, making it adaptable to various educational contexts and capable of being expanded with additional features as needed. This flexibility ensures that CreatAIlity can be easily integrated into different curricula and used with diverse age groups, enhancing its utility as an educational tool.

5.2 Contributions to the Field

CreatAIlity also contributes to the fields of educational technology and natural language processing. By bridging the gap between the creative potential of AI and the structured requirements of educational storytelling, this framework sets a new standard for how AI can be used to support learning and creativity in children. It also provides a practical example of how LLMs can be harnessed in a controlled manner, addressing the common challenge of non-determinism in AI outputs.

The comprehensive evaluation of CreatAIlity, utilizing tools such as the System Usability Scale (SUS), Game Experience Questionnaire (GEQ), and NASA Task Load Index (NASA-TLX), has yielded valuable insights into the framework's usability, effectiveness, and overall user experience. The positive outcomes from these assessments affirm the framework's potential to innovate in educational practices by making learning more engaging and interactive. These encouraging results motivate further studies and explorations in this area, highlighting the framework's promise in enhancing educational methodologies.

5.3 Future Research Directions

While this thesis has laid a foundation, there are several avenues for future research that could further enhance the framework. One area of interest is the refinement of techniques for managing the non-deterministic nature of LLMs. Future work could explore different models or fine-tuning strategies to improve consistency in narrative outputs, ensuring that the stories generated are not only creative but also aligned with educational goals.

Additionally, the scalability of CreatAIlity could be further investigated. While the current design supports modularity, understanding how the framework performs under varying conditions - such as different user loads or more complex narratives - could lead to improvements in performance and efficiency. This might involve optimizing the underlying algorithms or exploring cloud-based solutions to support larger-scale deployments.

Another promising direction is the exploration of broader applications for the framework. Beyond educational settings, CreatAIlity could be adapted for use in entertainment, therapy, or even corporate training environments where guided storytelling could play a role in skill development or team-building exercises.

5.4 Potential Impact

The impact of CreatAIlity on educational practices could be profound. By providing a tool that enhances creativity while ensuring educational value, this framework has the potential to improve, or event transform, how narrative skills are taught and nurtured in children. Its adaptability means that it can be tailored to various learning environments, making it a versatile addition to educational technology.

Furthermore, CreatAIlity's approach to combining AI with structured learning could inspire new methodologies in both AI development and education. As AI continues to evolve, frameworks like CreatAIlity could pave the way for more sophisticated educational tools that leverage the strengths of AI while mitigating its limitations.

5.5 Practical Applications

In addition to its educational potential, CreatAIlity could find applications in other fields where guided storytelling is valuable. For instance, in therapeutic settings, storytelling is often used as a means of expression and processing emotions. CreatAIlity could be adapted to help individuals create narratives that explore their feelings in a controlled and supportive environment. Similarly, in the entertainment industry, the framework could be used to develop interactive storytelling experiences that are both engaging and structured, offering users a unique blend of creativity and narrative coherence.

5.6 Final Notes

In conclusion, the CreatAIlity framework represents a significant step forward in integrating artificial intelligence with educational technology, offering a novel approach to fostering creativity in children through guided storytelling. Beyond its technical design and implementation, the framework's modularity, flexibility, and extensibility are intended to allow adaptation across a wide range of educational contexts and to be expanded for broader applications, while also serving as a platform for exploring new frontiers in this field.

As AI, particularly Generative AI, remains in its early stages, with their power evident but their potential applications still being explored, CreatAIlity makes a meaningful advance in this direction. Ultimately, it aspires to contribute more than just a tool - it serves as an exploratory venture into discovering new and effective ways to harness the capabilities of AI for educational and creative purposes.

Appendix A

CreatAIlity Framework and PoC Application Code Metrics

A.1 Raw Metrics

Metric	Value
Total Lines of Code (LOC)	3551
Logical Lines of Code (LLOC)	2095
Source Lines of Code (SLOC)	2751
Comments	101
Single-line Comments	84
Multi-line Comments	0
Blank Lines	716
Comment Stats	
(Comments per LOC)	3%
(Comments per SLOC)	4%
(Comments + Multi-line Comments per LOC)	3%

TABLE A.1: Raw Metrics Summary

A.2 Cyclomatic Complexity

Score Range	Rank	Risk and Description
1 - 5	A	Low risk - simple block
6 - 10	B	Low risk - well-structured and stable block
11 - 20	C	Moderate risk - slightly complex block
21 - 30	D	More than moderate risk - more complex block
31 - 40	E	High risk - complex block, alarming
41+	F	Very high risk - error-prone, unstable block

TABLE A.2: Cyclomatic Complexity Reference Risk Levels (Lacchia, n.d.)

Class	Cyclomatic Complexity
OutputHandlerStreamingEventMessage	A (2)
OutputHandlerStreamingCallbackHandler	A (2)
OutputHandlerStreamingEventType	A (1)
TkinterInputHandler	A (3)
InputHandlerEventMessage	A (2)
InputHandlerInterface	A (2)
ConsoleInputHandler	A (2)
WebSocketInputHandler	A (2)
InputHandlerEventType	A (1)
StackTkinterOutputHandler	A (5)
ConsoleOutputHandler	A (4)
TkinterOutputHandler	A (4)
WebSocketOutputHandler	A (4)
StackWebSocketOutputHandler	A (4)
OutputHandlerEventMessage	A (2)
OutputHandlerInterface	A (2)
OutputHandlerEventType	A (1)
LLMCallEventMessage	A (2)
ChatEventMessage	A (2)
ChatEventResponse	A (2)
BaseLLMHandler	A (2)
LLMCallEventType	A (1)
ChatEventType	A (1)
TGIHandler	A (4)
OllamaHandler	A (4)
EventBroker	A (5)
EventType	A (1)
EventMessage	A (1)
EventResponse	A (1)
StateManager	A (2)
AppConfig	A (3)
StateMachineController	A (2)
StatePersistenceEventMessage	A (4)
StatePersistenceManager	A (2)
ResourceManager	A (2)
DebugHandler	A (2)
BaseAppNamespace	A (2)
AppCustomJsonWrapper	A (3)
Resource	A (4)
ResourceEncoder	A (3)
ResourceDecoder	A (3)
ResourceAction	A (3)
ResourceEventMessage	A (2)
ResourceEventType	A (1)
StateMachine	A (3)
StateMachineEventMessage	A (2)
StateMachineEventType	A (1)
StateConfig	A (2)
State	A (2)

TABLE A.3: Cyclomatic Complexity Scores for Classes

A.3 Maintainability Index

MI Score	Rank	Maintainability
100 - 20	A	Very high
19 - 10	B	Medium
9 - 0	C	Extremely low

TABLE A.4: Maintainability Index Reference Table (Lacchia, n.d.)

File	Maintainability Index
lib/io/streaming_outputs.py	A (60.26)
lib/io/inputs.py	A (48.37)
lib/io/outputs.py	A (39.13)
lib/log/app_logging.py	A (72.53)
lib/llm/llm_handler.py	A (48.48)
lib/llm/huggingface/tgi.py	A (45.89)
lib/llm/langchain/ollama.py	A (48.50)
lib/app/event_broker.py	A (49.26)
lib/app/state_manager.py	A (61.27)
lib/app/app_config.py	A (73.00)
lib/app/manager/state_machine_controller.py	A (60.34)
lib/app/manager/state_persistence_manager.py	A (38.20)
lib/app/manager/resource_manager.py	A (74.02)
lib/app/manager/debug_handler.py	A (100.00)
lib/app/flask_socketio_app/base_app_namespace.py	A (53.97)
lib/app/flask_socketio_app/flask_io_json_wrapper.py	A (67.02)
lib/sm/resource.py	A (40.88)
lib/sm/state_machine.py	A (32.84)
lib/sm/state.py	A (100.00)
app/story_builder/websocket/story_builder_websocket.py	A (67.03)
app/story_builder/console/story_builder_console_ui.py	A (69.45)
app/story_builder/resources/update_handlers.py	A (61.43)
app/story_builder/tk/story_builder_tk_gui.py	A (63.29)
app/tk/ui/ui_components.py	A (52.03)
app/tk/ui/debug_window.py	A (100.00)
app/utils/helpers.py	A (63.52)

TABLE A.5: Maintainability Index Scores for CreatAIlity and PoC Applications

Appendix B

Evaluation Questionnaires

B.1 System Usability Scale (SUS)

No.	Statement
1	I think that I would like to use this system frequently.
2	I found the system unnecessarily complex.
3	I thought the system was easy to use.
4	I think that I would need the support of a technical person to be able to use this system.
5	I found the various functions in this system were well integrated.
6	I thought there was too much inconsistency in this system.
7	I would imagine that most people would learn to use this system very quickly.
8	I found the system very cumbersome to use.
9	I felt very confident using the system.
10	I needed to learn a lot of things before I could get going with this system.

TABLE B.1: System Usability Scale Evaluation Statements

Value	Answers
1	Strongly agree
2	Agree
3	Neutral
4	Disagree
5	Strongly disagree

TABLE B.2: System Usability Scale Evaluation Answers

B.2 Game Experience Questionnaire (GEQ)

No.	Statement	No.	Statement
1	I felt content	18	I felt imaginative
2	I felt skilful	19	I felt that I could explore things
3	I was interested in the game's story	20	I enjoyed it
4	I thought it was fun	21	I was fast at reaching the game's targets
5	I was fully occupied with the game	22	I felt annoyed
6	I felt happy	23	I felt pressured
7	It gave me a bad mood	24	I felt irritable
8	I thought about other things	25	I lost track of time
9	I found it tiresome	26	I felt challenged
10	I felt competent	27	I found it impressive
11	I thought it was hard	28	I was deeply concentrated in the game
12	It was aesthetically pleasing	29	I felt frustrated
13	I forgot everything around me	30	It felt like a rich experience
14	I felt good	31	I lost connection with the outside world
15	I was good at it	32	I felt time pressure
16	I felt bored	33	I had to put a lot of effort into it
17	I felt successful		

TABLE B.3: Game Experience Questionnaire Evaluation Statements

Value	Answers
1	not at all
2	slightly
3	moderately
4	fairly
5	extremely

TABLE B.4: Game Experience Questionnaire Evaluation Answers

B.3 NASA Task Load Index (NASA-TLX)

No.	Question
1	Mental Demand - How mentally demanding was the task?
2	Physical Demand - How physically demanding was the task?
3	Temporal Demand - How hurried or rushed was the pace of the task?
4	Performance - How successful were you in accomplishing what you were asked to do?
5	Effort - How hard did you have to work to accomplish your level of performance?
6	Frustration - How insecure, discouraged, irritated, stressed, and annoyed were you?

TABLE B.5: NASA Task Load Index Evaluation Questions

Value	**Answers**
1	Very Low
2	
3	
4	
5	Very High

TABLE B.6: NASA Task Load Index Evaluation Answers

Bibliography

Addone, Agnese, Giuseppina Palmieri, and Maria Angela Pellegrino (2022). "Engaging Children in Digital Storytelling". en. In: *Methodologies and Intelligent Systems for Technology Enhanced Learning, 11th International Conference*. Ed. by Fernando De la Prieta et al. Cham: Springer International Publishing, pp. 261–270. ISBN: 978-3-030-86618-1. DOI: 10.1007/978-3-030-86618-1_26.

Alagar, V.S. and K. Periyasamy (2011). *Specification of Software Systems*. en. Texts in Computer Science. London: Springer London. ISBN: 978-0-85729-276-6 978-0-85729-277-3. DOI: 10.1007/978-0-85729-277-3. URL: http://link.springer.com/10.1007/978-0-85729-277-3 (visited on 06/24/2024).

baeldung (Oct. 2021). *Statically Typed vs Dynamically Typed Languages | Baeldung on Computer Science*. en-US. URL: https://www.baeldung.com/cs/statically-vs-dynamically-typed-languages (visited on 09/01/2024).

Brooke, john (1996). "SUS: A 'Quick and Dirty' Usability Scale". In: *Usability Evaluation In Industry*. Num Pages: 6. CRC Press. ISBN: 978-0-429-15701-1.

Cai, Carrie J., Shamsi T. Iqbal, and Jaime Teevan (May 2016). "Chain Reactions: The Impact of Order on Microtask Chains". en. In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. San Jose California USA: ACM, pp. 3143–3154. ISBN: 978-1-4503-3362-7. DOI: 10.1145/2858036.2858237. URL: https://dl.acm.org/doi/10.1145/2858036.2858237 (visited on 06/24/2024).

Catala, Alejandro, Hannie Gijlers, and Iris Visser (Apr. 2023). "Guidance in storytelling tables supports emotional development in kindergartners". en. In: *Multimedia Tools and Applications* 82.9, pp. 12907–12937. ISSN: 1380-7501, 1573-7721. DOI: 10.1007/s11042-022-14049-7. URL: https://link.springer.com/10.1007/s11042-022-14049-7 (visited on 07/10/2024).

Chen, Lijia, Pingping Chen, and Zhijian Lin (2020). "Artificial Intelligence in Education: A Review". In: *IEEE Access* 8. Conference Name: IEEE Access, pp. 75264–75278. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2988510. URL: https://ieeexplore.ieee.org/document/9069875 (visited on 07/09/2024).

Chilton, Lydia B, James A Landay, and Daniel S Weld (2016). "HumorTools: A Microtask Workflow for Writing News Satire". en. In: *El Paso, Texas: ACM*.

Colledanchise, Michele and Petter Ögren (July 2018). *Behavior Trees in Robotics and AI: An Introduction*. arXiv:1709.00084 [cs]. DOI: 10.1201/9780429489105. URL: http://arxiv.org/abs/1709.00084 (visited on 09/01/2024).

Ehrlinger, Lisa and Wolfram Wöß (2016). "Towards a Definition of Knowledge Graphs". en. In: *SEMANTiCS (Posters, Demos, SuCCESS)* 48.1-4, p. 2.

Garzotto, Franca, Paolo Paolini, and Amalia Sabiescu (June 2010). "Interactive storytelling for children". en. In: *Proceedings of the 9th International Conference on Interaction Design and Children*. Barcelona Spain: ACM, pp. 356–359. ISBN: 978-1-60558-951-0. DOI: 10.1145/1810543.1810613. URL: https://dl.acm.org/doi/10.1145/1810543.1810613 (visited on 07/02/2024).

Gmeiner, Frederic and Nur Yildirim (2023). "Dimensions for Designing LLM-based Writing Support". en. In: *In2Writing Workshop at CHI*.

Hart, Sandra G. (Oct. 2006). “Nasa-Task Load Index (NASA-TLX); 20 Years Later”. en. In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 50.9. Publisher: SAGE Publications Inc, pp. 904–908. ISSN: 1071-1813. DOI: 10.1177/154193120605000909. URL: https://doi.org/10.1177/154193120605000909 (visited on 09/01/2024).

Hart, Sandra G. and Lowell E. Staveland (1988). “Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research”. en. In: *Advances in Psychology*. Vol. 52. Elsevier, pp. 139–183. ISBN: 978-0-444-70388-0. DOI: 10.1016/S0166-4115(08)62386-9. URL: https://linkinghub.elsevier.com/retrieve/pii/S0166411508623869 (visited on 09/01/2024).

Hibbin, Rebecca (Oct. 2016). “The psychosocial benefits of oral storytelling in school: developing identity and empathy through narrative”. en. In: *Pastoral Care in Education* 34.4, pp. 218–231. ISSN: 0264-3944, 1468-0122. DOI: 10.1080/02643944.2016.1225315. URL: https://www.tandfonline.com/doi/full/10.1080/02643944.2016.1225315 (visited on 07/10/2024).

Jones Bartoli, Alice (Nov. 2018). *Using storytelling to promote literacy, communication and socio-emotional development in the early years*. eng. Report. Num Pages: 21. Tales Toolkit. URL: https://research.gold.ac.uk/id/eprint/24937/ (visited on 07/10/2024).

Kooli, Chokri (Jan. 2023). “Chatbots in Education and Research: A Critical Examination of Ethical Implications and Solutions”. en. In: *Sustainability* 15.7. Number: 7 Publisher: Multidisciplinary Digital Publishing Institute, p. 5614. ISSN: 2071-1050. DOI: 10.3390/su15075614. URL: https://www.mdpi.com/2071-1050/15/7/5614 (visited on 07/09/2024).

Lacchia, Michele (n.d.). *Welcome to Radon's documentation! — Radon 4.1.0 documentation*. URL: https://radon.readthedocs.io/en/latest/ (visited on 09/01/2024).

Lee, Mina, Percy Liang, and Qian Yang (Apr. 2022). “CoAuthor: Designing a Human-AI Collaborative Writing Dataset for Exploring Language Model Capabilities”. en. In: *CHI Conference on Human Factors in Computing Systems*. arXiv:2201.06796 [cs], pp. 1–19. DOI: 10.1145/3491102.3502030. URL: http://arxiv.org/abs/2201.06796 (visited on 06/24/2024).

Lewis, James (Mar. 2018). “The System Usability Scale: Past, Present, and Future”. In: *International Journal of Human-Computer Interaction*, pp. 1–14. DOI: 10.1080/10447318.2018.1455307.

Lin, Zhicheng (Mar. 2024). “How to write effective prompts for large language models”. en. In: *Nature Human Behaviour* 8.4, pp. 611–615. ISSN: 2397-3374. DOI: 10.1038/s41562-024-01847-2. URL: https://www.nature.com/articles/s41562-024-01847-2 (visited on 07/02/2024).

Liu, Tongliang et al., eds. (2024). *AI 2023: Advances in Artificial Intelligence: 36th Australasian Joint Conference on Artificial Intelligence, AI 2023, Brisbane, QLD, Australia, November 28–December 1, 2023, Proceedings, Part II*. en. Vol. 14472. Lecture Notes in Computer Science. Singapore: Springer Nature Singapore. ISBN: 978-981-9983-90-2 978-981-9983-91-9. DOI: 10.1007/978-981-99-8391-9. URL: https://link.springer.com/10.1007/978-981-99-8391-9 (visited on 06/24/2024).

Lu, Zhuoran et al. (May 2024). *Corporate Communication Companion (CCC): An LLM-empowered Writing Assistant for Workplace Social Media*. en. arXiv:2405.04656 [cs]. URL: http://arxiv.org/abs/2405.04656 (visited on 06/24/2024).

Mirowski, Piotr et al. (Apr. 2023). “Co-Writing Screenplays and Theatre Scripts with Language Models: Evaluation by Industry Professionals”. en. In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Hamburg Germany:

ACM, pp. 1–34. ISBN: 978-1-4503-9421-5. DOI: 10.1145/3544548.3581225. URL: https://dl.acm.org/doi/10.1145/3544548.3581225 (visited on 06/24/2024).

Mishra, Abhijit et al. (2019). "Storytelling from Structured Data and Knowledge Graphs : An NLG Perspective". en. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*. Florence, Italy: Association for Computational Linguistics, pp. 43–48. DOI: 10.18653/v1/P19-4009. URL: https://www.aclweb.org/anthology/P19-4009 (visited on 06/24/2024).

Nack, Frank and Andrew S. Gordon, eds. (2016). *Interactive Storytelling: 9th International Conference on Interactive Digital Storytelling, ICIDS 2016, Los Angeles, CA, USA, November 15–18, 2016, Proceedings*. en. Vol. 10045. Lecture Notes in Computer Science. Cham: Springer International Publishing. ISBN: 978-3-319-48278-1 978-3-319-48279-8. DOI: 10.1007/978-3-319-48279-8. URL: https://link.springer.com/10.1007/978-3-319-48279-8 (visited on 06/24/2024).

OpenAI Platform (n.d.). en. URL: https://platform.openai.com/docs/guides/text-generation/reproducible-outputs (visited on 07/09/2024).

Ouyang, Shuyin et al. (Aug. 2023). *LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation*. en. arXiv:2308.02828 [cs]. URL: http://arxiv.org/abs/2308.02828 (visited on 06/24/2024).

Pimentel, João et al. (2019). "Creating Modelling Tools for i* Language Extensions." In: *12th International iStar Workshop*. URL: http://ceur-ws.org/Vol-2490/paper16.pdf.

Poels, K., Y.A.W. de Kort, and W.A. IJsselsteijn (2007). *D3.3 : Game Experience Questionnaire: development of a self-report measure to assess the psychological impact of digital games*. Eindhoven: Technische Universiteit Eindhoven.

Renzi, Gianluigi et al. (Aug. 2023). "A storytelling framework based on multimedia knowledge graph using linked open data and deep neural networks". en. In: *Multimedia Tools and Applications* 82.20, pp. 31625–31639. ISSN: 1380-7501, 1573-7721. DOI: 10.1007/s11042-023-14398-x. URL: https://link.springer.com/10.1007/s11042-023-14398-x (visited on 06/24/2024).

Riach, Duncan (2019). "Determinism in Deep Learning (S9911)". en. In: *GTC 2019*.

Sawyer, R. Keith, ed. (2003). *Creativity and development*. en. Counterpoints. New York: Oxford University Press. ISBN: 978-0-19-514899-2 978-0-19-514900-5.

Sklyarov, V. (June 1999). "Hierarchical finite-state machines and their use for digital control". en. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 7.2, pp. 222–228. ISSN: 1063-8210, 1557-9999. DOI: 10.1109/92.766749. URL: http://ieeexplore.ieee.org/document/766749/ (visited on 06/28/2024).

Tao, Yong, Hongxing Wei, and Tianmiao Wang (2008). "A Speech Interaction System Based on Finite State Machine for Service Robot". en. In: *2008 International Conference on Computer Science and Software Engineering*. Wuhan, China: IEEE, pp. 1111–1114. ISBN: 978-0-7695-3336-0. DOI: 10.1109/CSSE.2008.627. URL: http://ieeexplore.ieee.org/document/4721947/ (visited on 06/24/2024).

Teevan, Jaime et al. (May 2016). "Productivity Decomposed: Getting Big Things Done with Little Microtasks". en. In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. San Jose California USA: ACM, pp. 3500–3507. ISBN: 978-1-4503-4082-3. DOI: 10.1145/2851581.2856480. URL: https://dl.acm.org/doi/10.1145/2851581.2856480 (visited on 06/24/2024).

Tratt, Laurence (Jan. 2009). "Chapter 5 Dynamically Typed Languages". In: *Advances in Computers*. Vol. 77. Elsevier, pp. 149–184. DOI: 10.1016/S0065-2458(09)01205-4. URL: https://www.sciencedirect.com/science/article/pii/S0065245809012054 (visited on 09/01/2024).

Wu, Tongshuang, Michael Terry, and Carrie J. Cai (Mar. 2022). *AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts*. arXiv:2110.01691 [cs]. DOI: 10.48550/arXiv.2110.01691. URL: http://arxiv.org/abs/2110.01691 (visited on 06/24/2024).

Yang, Xinran and Ilaria Tiddi (2020). "Creative Storytelling with Language Models and Knowledge Graphs". en. In.