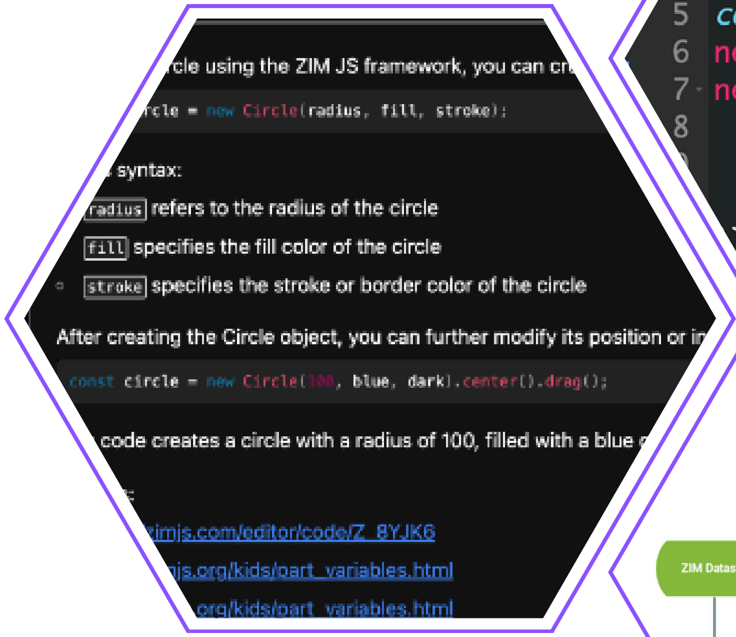
ZIMJS AI Assistant

Enhancing User Experience with Machine Learning



rcle using the ZIM JS framework, you can cr

```
rcle = new Circle(radius, fill, stroke);
```

syntax:

- radius refers to the radius of the circle
- fill specifies the fill color of the circle
- stroke specifies the stroke or border color of the circle

After creating the Circle object, you can further modify its position or in

```
const circle = new Circle(100, blue, dark).center().drag();
```

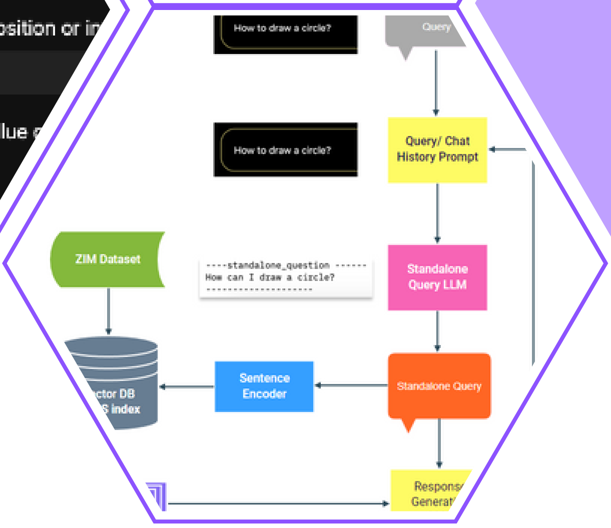
code creates a circle with a radius of 100, filled with a blue c

zimjs.com/editor/code/Z_8YJK6

js.org/kids/part_variables.html

org/kids/part_variables.html

```
/ given F (Frame), S
F.color = lighter;

const circle = new Circle(8
new Circle(22, red).addTo(ci
new MotionController({
    type:"mousemove",
    target:circle,
    flip:"both"
});
```



Prepared by
Suha Islaih
suha@mcode.club
647-901-8305

Date
Sept 7, 2024

Table Of Content

1. Executive Summary

The ZIMJS AI Assistant project was initiated to address the challenges faced by users on the zimjs.com website in finding quick, accurate answers to their coding-related questions. This report outlines the journey from problem identification to the deployment of a machine learning-based chatbot solution. The project leveraged advanced NLP techniques, including Retrieval-Augmented Generation (RAG) and the GPT-3.5 Turbo model, to enhance user experience by delivering context-specific responses. Key achievements include the integration of a user-friendly interface built with Gradio, successful deployment using Hugging Face Spaces, and iterative improvements based on user feedback. The report concludes with a discussion on future work and the potential for applying this solution to other domains.

2. Introduction

BACKGROUND

The zimjs.com platform serves as a valuable resource for developers using the ZIM JavaScript library to create animations, games, and interactive applications. Despite the comprehensive documentation and active community forum, users often faced difficulties in finding relevant answers quickly. This gap led to user frustration and hindered overall engagement.

PROJECT GOAL

The primary objective of this project was to develop an AI-powered chatbot that could accurately interpret user queries and provide relevant answers, particularly for code-related questions. This would not only improve user satisfaction but also enhance the overall usability of the zimjs.com platform. By automating the response process, the chatbot aims to reduce the load on human support while providing timely and accurate assistance to users. The chatbot is intended to serve as a first line of support, offering immediate help and directing users to appropriate resources when necessary.

3. Problem Statement

The primary problem faced by the zimjs.com community was the lack of an effective search or assistance tool that could interpret and respond to user queries efficiently. Manual search through forums and documentation slowed down the development process, making it difficult for users to get immediate help with their projects.

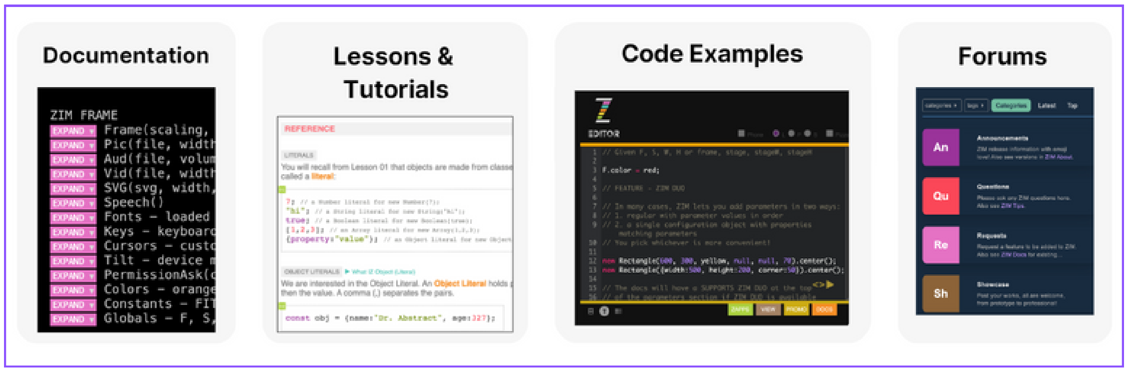
4. Methodology and Approach

• DATA COLLECTION AND LABELING

Data collection was conducted across several zimjs.com sources, including documentation, forum posts, tutorials, and code examples. To collect the necessary data for training the ZIMJS AI Assistant, a series of web scraping techniques were employed across various sections of the zimjs.com website. Tools like **Selenium** and **BeautifulSoup** were utilized to automate the extraction of code examples, tutorials, and interactive content from the site. This process involved navigating through web pages, identifying and extracting relevant JavaScript code snippets, and capturing associated metadata such as titles and URLs.

The extracted content was then cleaned, formatted, and organized into datasets suitable for training and fine-tuning machine learning models. This comprehensive data collection process provided a robust foundation for developing an AI assistant capable of delivering accurate and contextually relevant responses to user queries on the zimjs.com platform.

The data was then categorized into relevant labels to aid in model training. These categories included: Code examples, TutorialRequests and General Queries.
The data labeling process was critical to ensure that the chatbot could understand and classify incoming queries accurately. The collected data was tokenized using the appropriate tokenizer for each model, starting with **CodeBERT**.



```
ZIM FRAME
EXPAND ▾ Frame(scaling,
EXPAND ▾ Pic(file, width
EXPAND ▾ Aud(file, volum
EXPAND ▾ Vid(file, width
EXPAND ▾ SVG(svg, width,
EXPAND ▾ Speech()
EXPAND ▾ Fonts - loaded
EXPAND ▾ Keys - keyboard
EXPAND ▾ Cursors - custo
EXPAND ▾ Tilt - device m
EXPAND ▾ PermissionAsk(c
EXPAND ▾ Colors - orange
EXPAND ▾ Constants - FIT
EXPAND ▾ Globals - F, S,
```

You will recall from Lesson 01 that objects are made from classe called a literal:

```
7; // a Number literal for new Number(7);
"hi"; // a String literal for new String("hi");
true; // a Boolean literal for new Boolean(true);
[1,2,3]; // an Array literal for new Array(1,2,3);
{property:"value"}; // an Object literal for new Object
```

We are interested in the Object Literal. An Object Literal holds then the value. A comma (,) separates the pairs.

```
const obj = {name:"Dr. Abstract", age:327};
```

```
// Given F, S, W, H or frame, stage, stageW, stageH

F.color = red;

// FEATURE - ZIM DUO

// In many cases, ZIM lets you add parameters in two ways:
// 1. regular with parameter values in order
// 2. a single configuration object with properties
   matching parameters
// You pick whichever is more convenient!

new Rectangle(500, 300, yellow, null, null, 70).center();
new Rectangle({width:500, height:200, corner:50}).center();

// The docs will have a SUPPORTS ZIM DUO at the top
// of the parameters section if ZIM DUO is available
```

Image# 1: Comprehensive data from the ZIMJS website

• MODEL SELECTION AND TRAINING

The project initially explored fine-tuning pre-trained models such as CodeBERT and TinyLlama to handle the ZIMJS-specific queries. This involved preparing the dataset, tokenizing the data using model-specific tokenizers, and setting up the training process using appropriate libraries and frameworks. Despite these efforts, both models exhibited significant challenges in generating high-quality, contextually accurate responses, leading to the exploration of alternative approaches.

• CODEBERT(MICROSOFT/CODEBERT-BASE)

DATA PREPARATION

The ZIMJS dataset, comprising question-answer pairs, was reshaped into a suitable format for training CodeBERT. Tokenization was performed using the CodeBERT tokenizer, which combined ZIMJS code snippets with corresponding questions to create input sequences for the model. This process ensured that the model could effectively learn the relationship between user queries and appropriate code-related responses. For a dataset sample used for fine-tuning TinyLlama and CodeBERT, see Table 2 in Appendix B.

MODEL TRAINING

Using the RobertaForCausalLM model from Hugging Face, the training process was set up with the Trainer class. Multiple epochs were run, observing a decrease in training loss, indicating some level of learning. However, the performance remained suboptimal, with the model struggling to generate high-quality answers, as evidenced by relatively high loss metrics during both training and validation phases. The model showed limited ability to generalize beyond the training data, resulting in repetitive and contextually irrelevant responses

CHALLENGES

The primary issues with CodeBERT included overfitting and difficulty in generating contextually accurate and meaningful answers. The model's responses were often repetitive and failed to capture the nuances of the ZIMJS content, prompting a need to explore more effective models or approaches. Overfitting led to poor performance on unseen data, limiting the model’s practical utility in real-world scenarios.

• TINYLLAMA (TINYLLAMA-1.1B-QLORA-V2)

DATA PREPARATION

Similar to CodeBERT, the ZIMJS dataset was prepared and tokenized using TinyLlama’s tokenizer. A customized prompt format was applied to structure the data, ensuring compatibility with the model's input requirements. This involved formatting the questions and answers in a way that TinyLlama could process effectively, aiming to improve the model's understanding and generation capabilities.For a dataset sample used for fine-tuning TinyLlama and CodeBERT, see Table 2 in Appendix B.

MODEL TRAINING

TinyLlama-1.1B was fine-tuned using a low-rank adaptation (LoRA) approach with the SFTTrainer from the trl library. Hyperparameters such as batch size and learning rate were adjusted to improve training efficiency. Over multiple training steps, the loss gradually decreased, indicating improved learning. The LoRA approach helped in adapting the model to the specific nuances of the ZIMJS dataset without extensive computational resources.

CHALLENGES

Despite better training losses compared to CodeBERT, TinyLlama still struggled with generating relevant and non-repetitive content. The model often produced repetitive outputs lacking depth, which was insufficient to handle the diverse queries from the ZIMJS content effectively. These limitations led to the consideration of alternative models and approaches, as the generated responses did not meet the project's goals in terms of quality and relevance.

- ## DECISION TO PIVOT TO RAG

Given the limitations and challenges encountered with both CodeBERT and TinyLlama, the project pivoted to using Retrieval-Augmented Generation (**RAG**). **RAG** was chosen because it allows the model to retrieve relevant documents from a pre-built index (FAISS vector store) and then generate responses based on this retrieved information. This approach was expected to improve the accuracy and relevance of the chatbot's answers by leveraging specific, contextually relevant content from the ZIMJS dataset.

The **RAG** framework combines the strengths of information retrieval and generative models, enabling the chatbot to access a larger pool of information dynamically and generate more accurate and contextually appropriate responses. This method addresses the shortcomings of purely fine-tuned models by providing a mechanism to reference a comprehensive knowledge base during response generation.

The initial attempts with CodeBERT and TinyLlama provided valuable insights into the complexities of generating high-quality responses for the ZIMJS chatbot. While CodeBERT showed some learning progression, it failed to produce accurate and contextually relevant answers. TinyLlama improved upon this but still faced significant challenges, including repetitive outputs and limited depth in responses.

These experiences highlighted the need for a more robust approach that could handle the nuanced requirements of the ZIMJS content. The decision to move to RAG was based on the potential of combining retrieval mechanisms with generative models to enhance response quality.

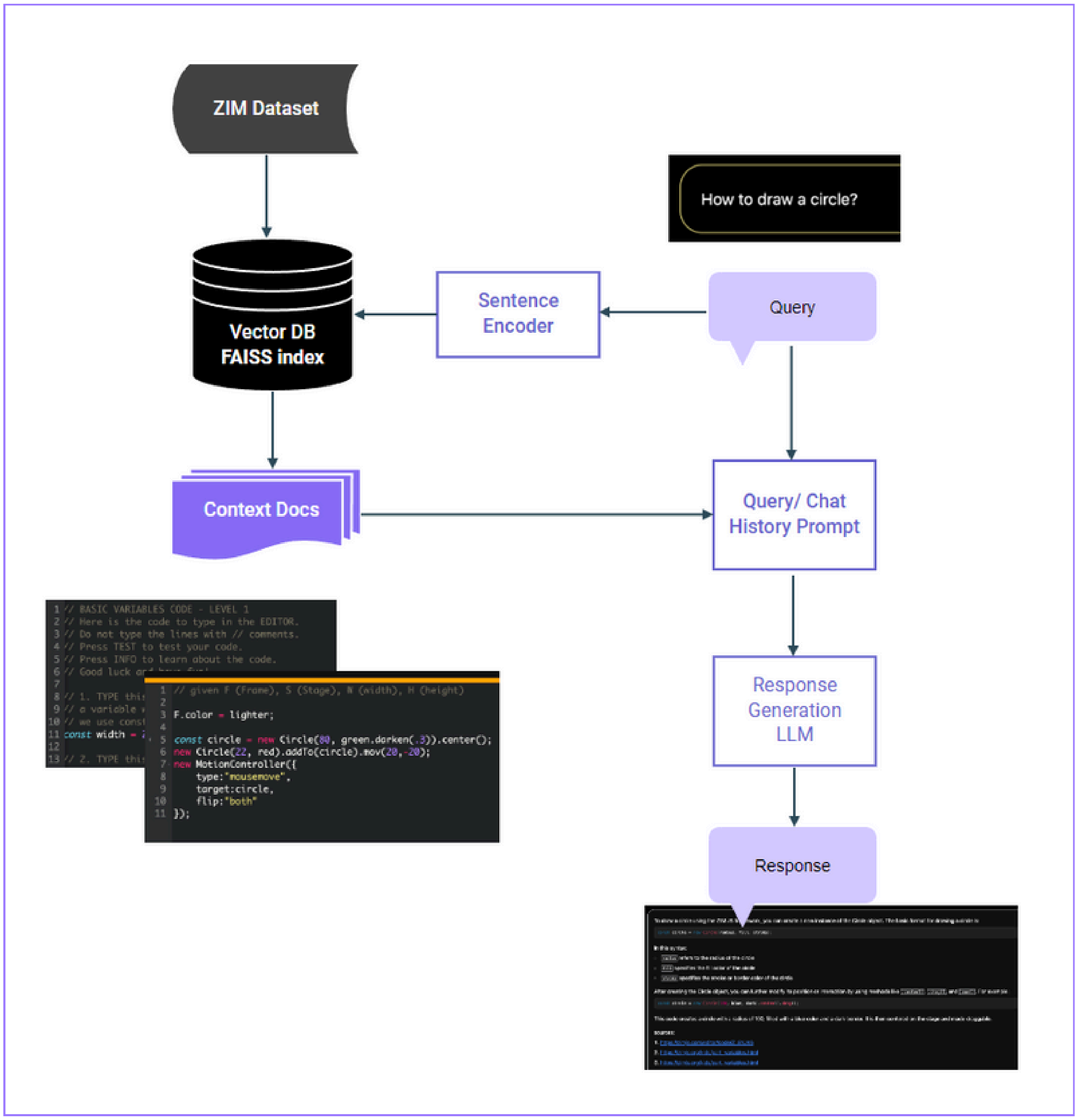
SEAMLESSLY UPDATING THE ZIMJS CHATBOT

One of the major advantages of using Retrieval-Augmented Generation (RAG) is the ease of updating the ZIMJS chatbot when new data becomes available. The process is straightforward, requiring minimal effort to refresh the dataset and integrate it into the chatbot without the need for re-training or model refinement. This flexibility allows the chatbot to stay current and responsive to new information with minimal overhead, making RAG an ideal solution for maintaining high-performance applications over time.

Basic RAG Workflow

Simple Retrieval-Augmented Generation Without Chat History Integration

The initial results of the RAG framework, focusing on generating contextually relevant responses, can be observed in the Basic RAG Workflow Figure #3. For an in-depth view of the Enhanced RAG with Standalone Question Generation process, including both single and multi-question workflows, please refer to Appendix D and Appendix respectively.



```
// BASIC VARIABLES CODE - LEVEL 1
// Here is the code to type in the EDITOR.
// Do not type the lines with // comments.
// Press TEST to test your code.
// Press INFO to learn about the code.
// Good luck or
 
// 1. TYPE thi
// a variable
// we use const
const width = 
 
// 2. TYPE thi
```

```
// given F (Frame), S (Stage), W (width), H (height)

F.color = lighter;

const circle = new Circle(80, green.darken(.3)).center();
new Circle(22, red).addTo(circle).mov(20,-20);
new MotionController({
    type:"mousemove",
    target:circle,
    flip:"both"
});
```

Figure 3: Basic RAG Workflow. (For Enhanced RAG with Standalone Question Generation, see Appendix D for the single question and Appendix E for multi-question workflows).

- IMPLEMENTATION OF MISTRAL-7B-INSTRUCT-V0.1 WITH RAG

MODEL SELECTION

The **Mistral-7B-Instruct-v0.1** model was selected for its strong performance in handling both coding and text-based queries. This model was loaded with quantization using **BitsAndBytesConfig**, enabling 4-bit precision loading to reduce memory usage and optimize GPU resource utilization. The quantization allowed the model to operate more efficiently on available hardware, making it feasible to handle larger volumes of queries without excessive computational overhead.

DATA PREPARATION AND VECTOR STORE CREATION

The zimjs.com dataset was processed by splitting the questions and answers into separate documents. These documents were then loaded into a FAISS vector store, enabling efficient retrieval of relevant context for each query. The FAISS index was built using HuggingFaceEmbeddings from the **sentence-transformers/all-mpnet-base-v2** model, optimizing retrieval accuracy for zimjs-specific content. This setup ensured that the chatbot could quickly access the most relevant pieces of information from the dataset, enhancing the quality and relevance of generated responses. For a dataset sample used with RAG and Mistral-7B-Instruct, refer to Table 3 in Appendix C.

INTEGRATION WITH LANGCHAIN

The RAG framework was implemented using LangChain, which combined the vector store for document retrieval with the Mistral model for response generation. Two main chains were created:

- **Retrieval Chain**: This chain fetched the most relevant documents from the **FAISS** index based on the user query. By retrieving pertinent information, the model could ground its responses in actual content from the ZIMJS dataset.
- **LLM Chain:** This chain utilized the Mistral model to generate responses based on the retrieved context. The generative model synthesized the information, producing coherent and contextually appropriate answers for the user.

CHATBOT TESTING AND PERFORMANCE

The chatbot was tested with various zimjs.com-related questions, and the RAG approach provided more contextually accurate and detailed answers compared to previous models like CodeBERT and TinyLlama. For example, queries related to specific code functionalities or troubleshooting common issues yielded more precise and helpful responses. However, despite the improved performance, the generated answers still required further refinement to align perfectly with the specific needs of zimjs users. Issues such as occasional irrelevant information and occasional template misuse persisted, indicating the need for continued optimization.

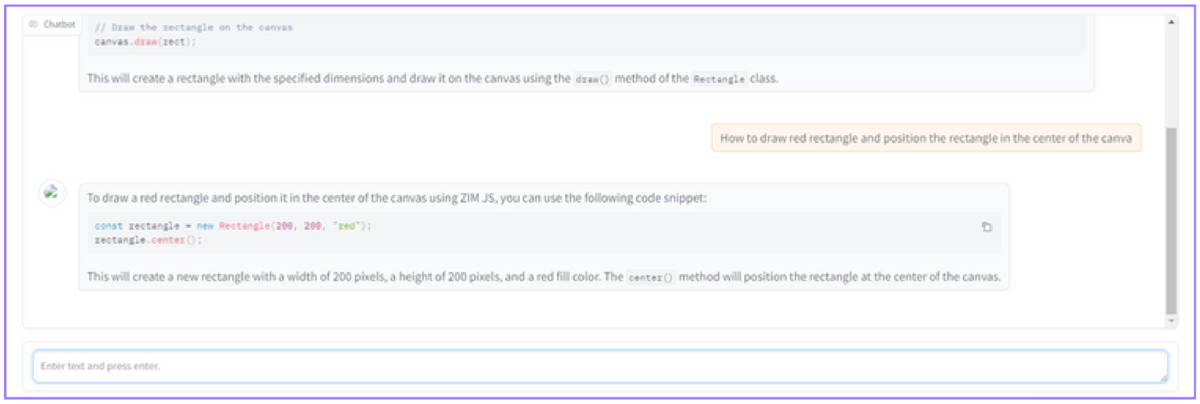
• FEEDBACK FROM DAN (ZIMJS.COM OWNER)

POSITIVE ASPECTS

Dan acknowledged that the RAG process was generating much better answers than a general-purpose model like GPT. He appreciated the attempts to generate context-specific answers that aligned more closely with zimjs content. The improvements in response accuracy and relevance were particularly noted as significant enhancements to the user experience.

AREAS FOR IMPROVEMENT

Dan pointed out that while the ZIM template was useful in a minority of cases (1/10 questions), in the majority of cases (9/10 questions), it was more misleading than helpful. This could be frustrating for students trying to use the chatbot, and he noted that more work was needed to refine the chatbot's output to avoid these issues. Specifically, the chatbot needed to enhance its context sensitivity to ensure responses were more aligned with user queries without unnecessary or misleading templating. This feedback was crucial in identifying the areas where the chatbot's responses could be further optimized to meet user expectations.



```
// Draw the rectangle on the canvas
canvas.draw(rect);
```

This will create a rectangle with the specified dimensions and draw it on the canvas using the `draw()` method of the `Rectangle` class.

How to draw red rectangle and position the rectangle in the center of the canva

To draw a red rectangle and position it in the center of the canvas using ZIM JS, you can use the following code snippet:

```
const rectangle = new Rectangle(200, 200, "red");
rectangle.center();
```

This will create a new rectangle with a width of 200 pixels, a height of 200 pixels, and a red fill color. The `center()` method will position the rectangle at the center of the canvas.

Image# 5: First ZIM chatbot DEMO

• SHIFT TO GPT-3.5 TURBO WITH RAG

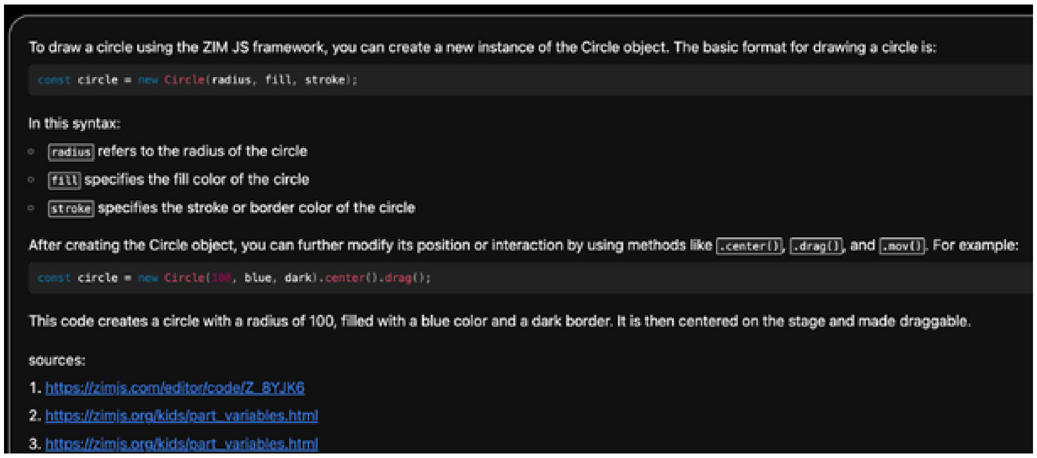
After assessing the high costs and complexity of deploying Mistral-7B, the project pivoted to GPT-3.5 Turbo. The model was deployed via the GPT API, which provided better cost-efficiency and allowed for easier integration with existing tools. This transition was further supported by a thorough evaluation of the datasets used, with a sample of the original dataset used with RAG and GPT-3.5-Turbo available for review in **Table 1** in Appendix A. For a dataset sample used with RAG and Mistral-7B-Instruct, refer to **Table 3** in Appendix C

REASONS FOR SHIFT

- **Cost-Effectiveness:** GPT-3.5 Turbo, accessible via API, eliminates the need for expensive cloud deployments and GPU resources, making it a more affordable and scalable solution.
- **Response Accuracy:** Ensuring the chatbot consistently provides accurate and relevant answers for a wide range of queries was an ongoing challenge. Balancing the retrieval and generation components to maintain high-quality responses across diverse topics required continuous optimization.
- **Resource Management:** While shifting to GPT-3.5 Turbo reduced deployment costs, managing the integration with RAG and ensuring efficient retrieval remained critical. Balancing cost, performance, and scalability was essential to maintaining an effective and sustainable chatbot solution.

INTEGRATION WITH GRADIO

- The chatbot's user interface was developed using Gradio, which provided an intuitive and responsive platform for users to interact with the chatbot. Additional features like chat history handling were implemented, allowing the model to save conversation history and generate standalone questions from previous queries.



To draw a circle using the ZIM JS framework, you can create a new instance of the Circle object. The basic format for drawing a circle is:

```
const circle = new Circle(radius, fill, stroke);
```

In this syntax:

- `radius` refers to the radius of the circle
- `fill` specifies the fill color of the circle
- `stroke` specifies the stroke or border color of the circle

After creating the Circle object, you can further modify its position or interaction by using methods like `.center()`, `.drag()`, and `.mov()`. For example:

```
const circle = new Circle(100, blue, dark).center().drag();
```

This code creates a circle with a radius of 100, filled with a blue color and a dark border. It is then centered on the stage and made draggable.

sources:

1. https://zimjs.com/editor/code/Z_8YJK6
2. https://zimjs.org/kids/part_variables.html
3. https://zimjs.org/kids/part_variables.html

Image# 6: Second ZIM chatbot DEMO

5. Results and Discussion

CHATBOT PERFORMANCE

The combination of GPT-3.5 Turbo and RAG produced significantly more accurate and contextually relevant answers compared to previous models. This improvement was confirmed through real-time testing and user feedback from Dan, the owner of zimjs.com.

DAN'S FEEDBACK AND ITERATION

Dan's feedback was crucial in guiding the iterative improvements of the chatbot. His insights helped identify specific areas where the chatbot's responses were lacking, allowing for targeted refinements to enhance the chatbot's utility and user satisfaction. By incorporating his feedback, the project was able to address critical issues and make meaningful improvements to the chatbot's performance.
For Dan's latest review & feedback, please refer to Appendix F

CHALLENGES AND FUTURE CONSIDERATIONS

Despite the improvements, the need for further refinement was evident, especially in generating more context-sensitive responses. Moving forward, the focus will be on fine-tuning the use of templates and expanding the chatbot's capabilities.

DEPLOYMENT AND SCALABILITY

Deployment on Hugging Face Spaces: The chatbot was deployed on Hugging Face Spaces for demonstration purposes. This platform was chosen for its ease of use, scalability, and ability to provide a seamless testing environment. It also allowed for public access, enabling a broader audience to test and provide feedback.

Cost-Efficiency: The use of the GPT-3.5 API significantly reduced the operational costs associated with cloud deployment, as it eliminated the need for dedicated cloud infrastructure and powerful GPUs.

6. Conclusion: Key Takeaways & Next Steps

The ZIMJS AI Assistant project demonstrates the effectiveness of combining retrieval-augmented generation with advanced language models to create a highly responsive and contextually aware chatbot. While initial models like CodeBERT and TinyLlama provided foundational insights, the pivot to RAG with GPT-3.5 Turbo marked a significant advancement in the chatbot's capabilities. The integration with Gradio facilitated a user-friendly interface, enhancing accessibility and usability. Continuous feedback and iterative improvements ensure that the chatbot remains a relevant and valuable tool for the ZIMJS community. Future work will focus on further refining response accuracy, expanding functionalities, and exploring broader applications of the developed framework.

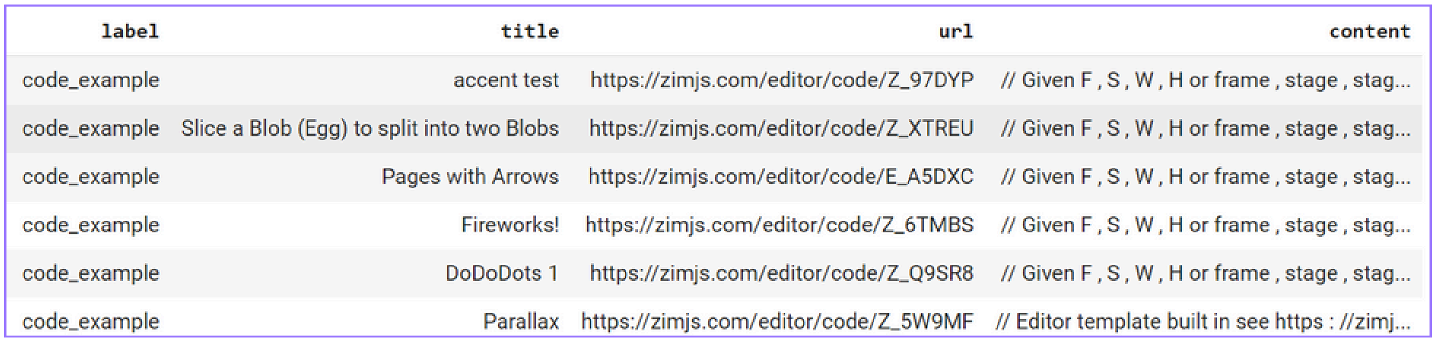
References

- **Hugging Face. (n.d.). Transformers Library. Retrieved from** https://huggingface.co/transformers/
- **OpenAI. (n.d.). GPT-3.5 Turbo. Retrieved from** https://openai.com/api/
- **Microsoft. (n.d.). CodeBERT. Retrieved from** https://github.com/microsoft/CodeBERT
- **TinyLlama. (n.d.). TinyLlama-1.1B-qlora-v2. Retrieved from** https://github.com/tinyllama
- **Facebook AI Research. (n.d.). FAISS: Facebook AI Similarity Search. Retrieved from** https://faiss.ai/
- **LangChain Documentation. (n.d.). Retrieved from** https://langchain.com/docs/
- **Gradio Documentation. (n.d.). Retrieved from** https://gradio.app/docs/
- **Sentence-Transformers. (n.d.). all-mpnet-base-v2. Retrieved from** https://huggingface.co/sentence-transformers/all-mpnet-base-v2
- **(Part 1) Build your own RAG with Mistral-7B and LangChain** https://medium.com/@thakermadhav/build-your-own-rag-with-mistral-7b-and-langchain-97d0c92fa146
- **(Part 2) Build a Conversational RAG with Mistral-7B and LangChain** https://medium.com/@thakermadhav/part-2-build-a-conversational-rag-with-langchain-and-mistral-7b-6a4ebe497185

Appendices

Appendix A

Original Dataset Sample used with RAG and GPT-3.5-Turbo

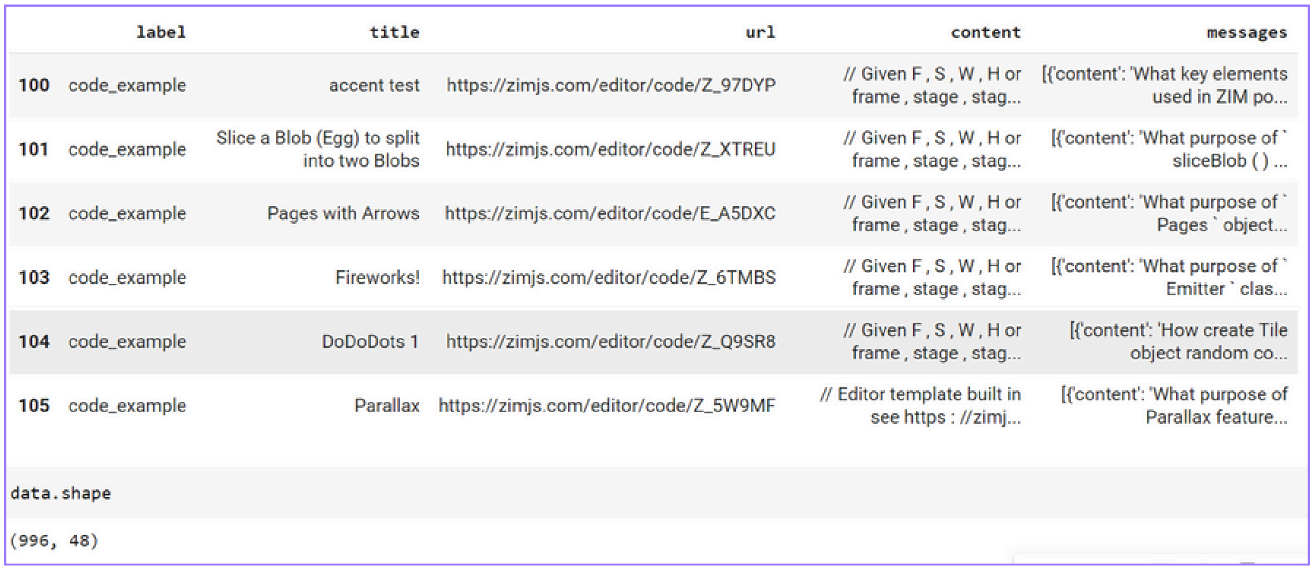


label	title	url	content
code_example	accent test	https://zimjs.com/editor/code/Z_97DYP	// Given F , S , W , H or frame , stage , stag...
code_example	Slice a Blob (Egg) to split into two Blobs	https://zimjs.com/editor/code/Z_XTREU	// Given F , S , W , H or frame , stage , stag...
code_example	Pages with Arrows	https://zimjs.com/editor/code/E_A5DXC	// Given F , S , W , H or frame , stage , stag...
code_example	Fireworks!	https://zimjs.com/editor/code/Z_6TMBS	// Given F , S , W , H or frame , stage , stag...
code_example	DoDoDots 1	https://zimjs.com/editor/code/Z_Q9SR8	// Given F , S , W , H or frame , stage , stag...
code_example	Parallax	https://zimjs.com/editor/code/Z_5W9MF	// Editor template built in see https : //zimj...

Table 1: Original Dataset Sample used with RAG and GPT-3.5-Turbo, including label, title, URLs, and content

Appendix B

Dataset Sample for Fine-Tuning TinyLlama and CodeBERT



	label	title	url	content	messages
100	code_example	accent test	https://zimjs.com/editor/code/Z_97DYP	// Given F , S , W , H or frame , stage , stag...	[{'content': 'What key elements used in ZIM po...
101	code_example	Slice a Blob (Egg) to split into two Blobs	https://zimjs.com/editor/code/Z_XTREU	// Given F , S , W , H or frame , stage , stag...	[{'content': 'What purpose of ` sliceBlob ( ) ...
102	code_example	Pages with Arrows	https://zimjs.com/editor/code/E_A5DXC	// Given F , S , W , H or frame , stage , stag...	[{'content': 'What purpose of ` Pages ` object...
103	code_example	Fireworks!	https://zimjs.com/editor/code/Z_6TMBS	// Given F , S , W , H or frame , stage , stag...	[{'content': 'What purpose of ` Emitter ` clas...
104	code_example	DoDoDots 1	https://zimjs.com/editor/code/Z_Q9SR8	// Given F , S , W , H or frame , stage , stag...	[{'content': 'How create Tile object random co...
105	code_example	Parallax	https://zimjs.com/editor/code/Z_5W9MF	// Editor template built in see https : //zimj...	[{'content': 'What purpose of Parallax feature...

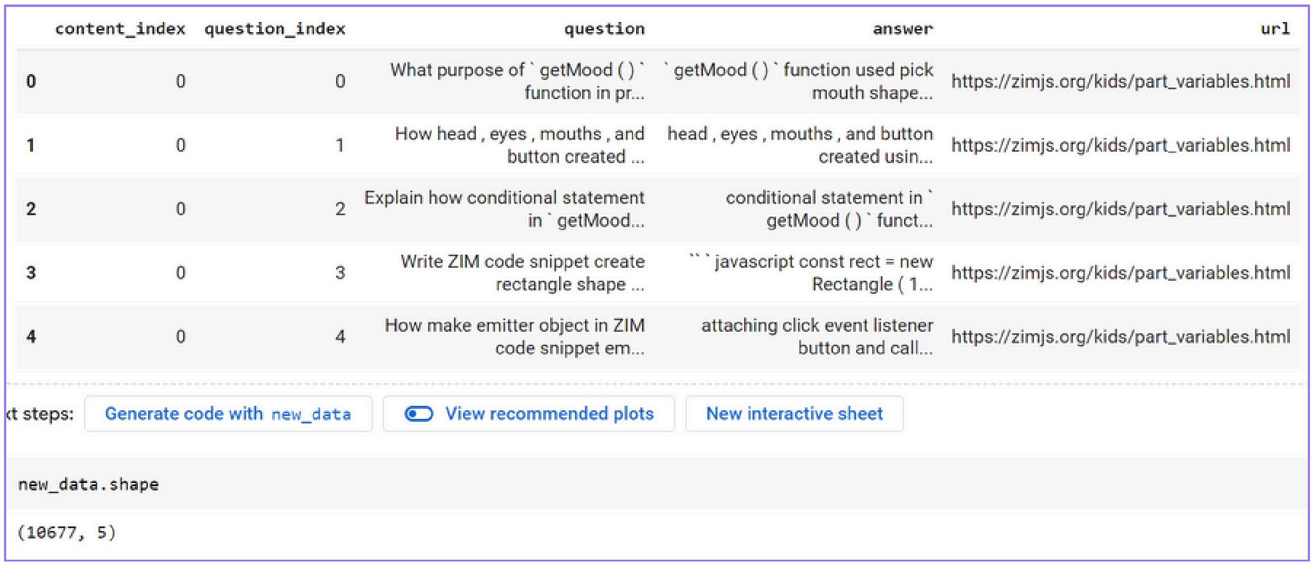
```
data.shape
```

```
(996, 48)
```

Table 2: Dataset Sample for Fine-Tuning TinyLlama and CodeBERT, focusing on general coding queries

Appendix C

Dataset Sample used with RAG and Mistral-7B-Instruct



	content_index	question_index	question	answer	url
0	0	0	What purpose of ` getMood ( ) ` function in pr...	` getMood ( ) ` function used pick mouth shape...	https://zimjs.org/kids/part_variables.html
1	0	1	How head , eyes , mouths , and button created ...	head , eyes , mouths , and button created usin...	https://zimjs.org/kids/part_variables.html
2	0	2	Explain how conditional statement in ` getMood...	conditional statement in ` getMood ( ) ` funct...	https://zimjs.org/kids/part_variables.html
3	0	3	Write ZIM code snippet create rectangle shape ...	``` javascript const rect = new Rectangle (1...	https://zimjs.org/kids/part_variables.html
4	0	4	How make emitter object in ZIM code snippet em...	attaching click event listener button and call...	https://zimjs.org/kids/part_variables.html

```
new_data.shape
```

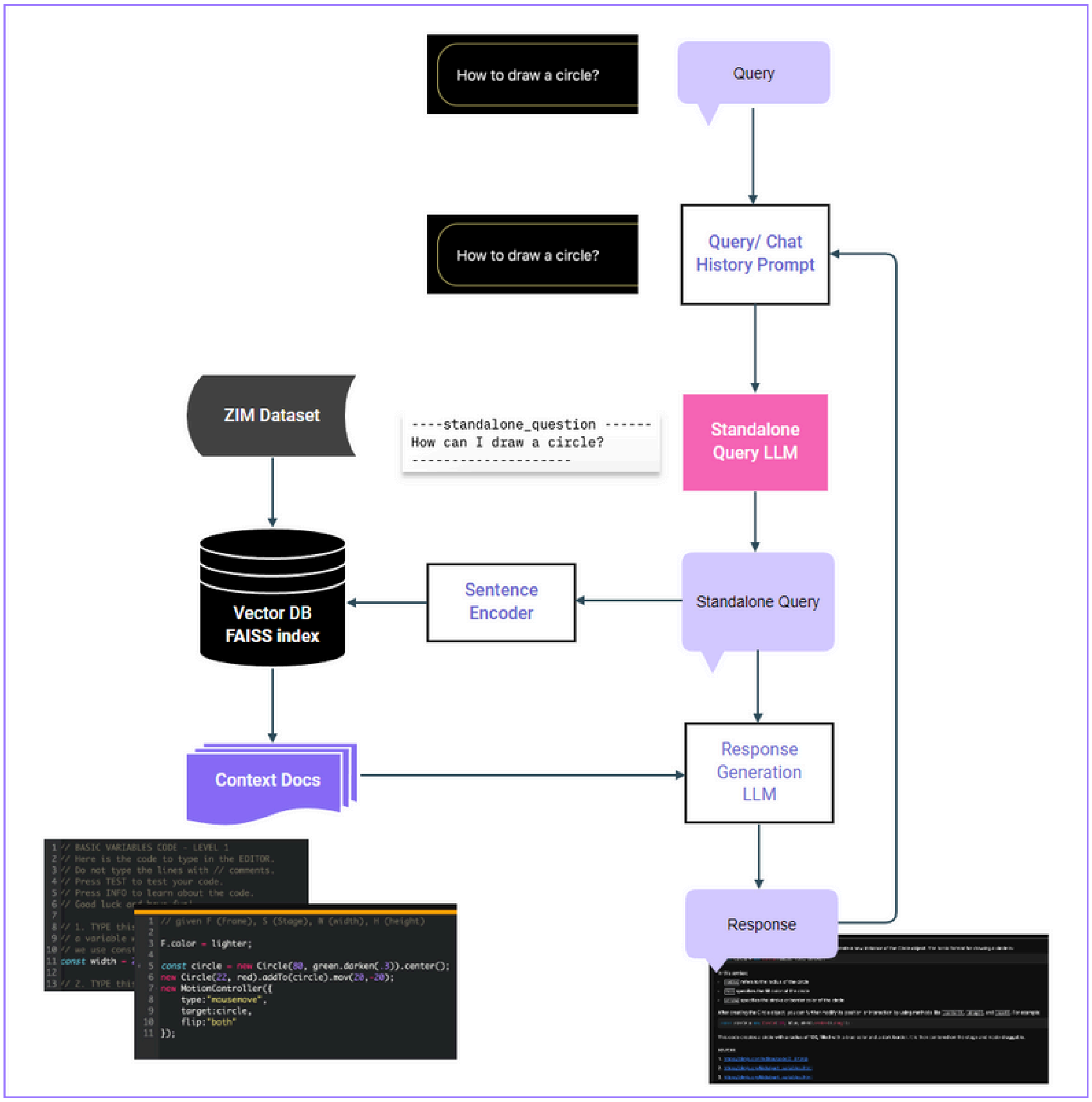
```
(10677, 5)
```

Table 3: Dataset Sample used with RAG and Mistral-7B-Instruct, illustrating its use in generating contextually relevant responses

Appendix D

Enhanced RAG with Standalone Question Generation

Incorporating Chat History to Generate Contextually Relevant Standalone Questions



```
----standalone_question ------
How can I draw a circle?
---------------------
```

```
// BASIC VARIABLES CODE - LEVEL 1
// Here is the code to type in the EDITOR.
// Do not type the lines with // comments.
// Press TEST to test your code.
// Press INFO to learn about the code.
// Good luck and have fun!

// 1. TYPE thi
// a variable
// we use const
const width = 

// 2. TYPE thi
```

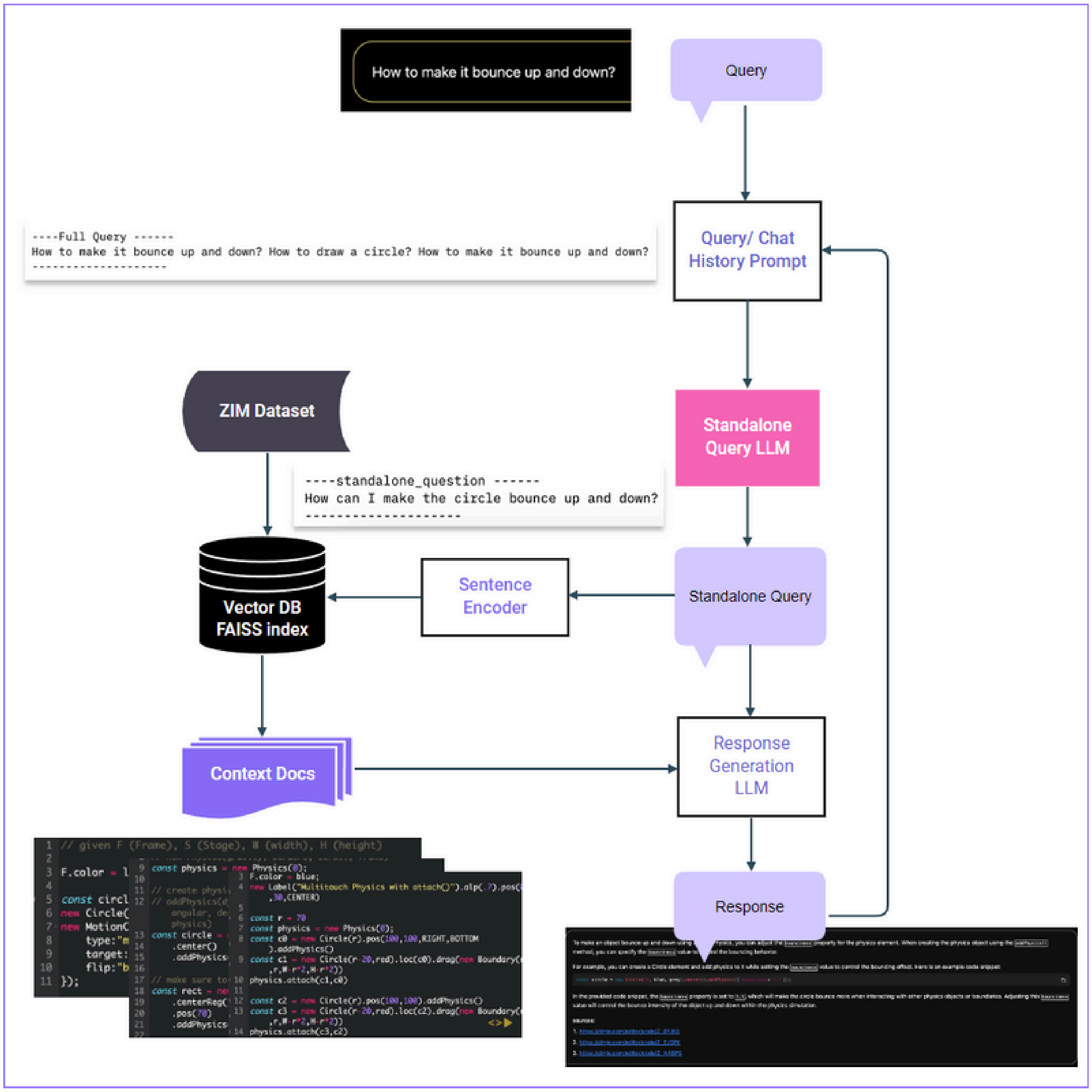
```
// given F (Frame), S (Stage), W (width), H (height)

F.color = lighter;

const circle = new Circle(80, green.darken(.3)).center();
new Circle(22, red).addTo(circle).mov(20,-20);
new MotionController({
    type:"mousemove",
    target:circle,
    flip:"both"
});
```

Image# 4: Enhanced RAG with Standalone Question Generation | One question

Appendix E

Enhanced RAG with Standalone Question Generation

Incorporating Chat History to Generate Contextually Relevant Standalone Questions



```
----Full Query ------
How to make it bounce up and down? How to draw a circle? How to make it bounce up and down?
--------------------
```

```
----standalone_question ------
How can I make the circle bounce up and down?
--------------------
```

```
// given F (Frame), S (Stage), W (width), H (height)

F.color = l

const circl
new Circle(
new MotionC
    type:"m
    target:
    flip:"b
});
```

```
const physics = new Physics(0);

// create physi
// addPhysics(d
    angular, de
    physics)
const circle = 
    .center()
    .addPhysics

// make sure to
const rect = ne
    .centerReg(
    .pos(70)
    .addPhysics
```

```
F.color = blue;
new Label("Multitouch Physics with attach()").alp(.7).pos(
    ,30,CENTER)

const r = 70
const physics = new Physics(0);
const c0 = new Circle(r).pos(100,100,RIGHT,BOTTOM
    ).addPhysics()
const c1 = new Circle(r-20,red).loc(c0).drag(new Boundary(
    ,r,W-r*2,H-r*2))
physics.attach(c1,c0)

const c2 = new Circle(r).pos(100,100).addPhysics()
const c3 = new Circle(r-20,red).loc(c2).drag(new Boundary(
    ,r,W-r*2,H-r*2))
physics.attach(c3,c2)
```

[illegible]

Sources:

1. [illegible]
2. [illegible]
3. [illegible]

Image# 4: Enhanced RAG with Standalone Question Generation | Two questions

Appendix F

Overall review from Dan

"Suha has done an excellent job collecting and organizing ZIM specific data for the final prompt to the LLM. We have gone through a series of tests and have greatly improved the results to a point where the answers are about 80% and hopefully, the minor issues will be detected by the user. This is well up from 0% when we tried general ChatGPT and about 5% when Suha first started. So we are going to implement the ZIM Chat Bot using a handy site that Suha set up."

Contact Suha

647-901-8305

suha@mcode.club

https://www.linkedin.com/in/suha-islaih/

About Suha Islaih

Suha Islaih is a dedicated technologist, co-founder, and program coordinator at My Code Club, where she leads initiatives that inspire over 600 students in coding and robotics. Her leadership has established key partnerships with organizations and government initiatives, underscoring her commitment to community and innovation in education.

Currently, Suha is advancing her knowledge through a Machine Learning Certificate program at York University, focusing on applying machine learning technologies in real-world scenarios. Her technical expertise includes both back-end and front-end development, with a strong proficiency in languages such as Python, PHP, MySQL, and JavaScript. Recognized for her communication and problem-solving skills, Suha excels in developing programs and leading events that drive technological innovation.

For more information on her work, visit her GitHub.