BLOOMING MINDS: AN ADAPTIVE E-LEARNING PLATFORM FOR INDIVIDUALS WITH DOWN SYNDROME

ID: 24-25J-248

B.Sc. (Hons) Degree in Information Technology

Sri Lanka Institute of Information Technology
Sri Lanka

April 2025

BLOOMING MINDS: AN ADAPTIVE E-LEARNING PLATFORM FOR INDIVIDUALS WITH DOWN SYNDROME

ID: 24-25J-248

Dissertation submitted in partial fulfillment of the requirements for the Bachelor of Science (Hons) in Information Technology

Sri Lanka Institute of Information Technology
Sri Lanka

April 2025

DECLARATION

I declare that this is my own work, and this dissertation does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. Also, I hereby grant to Sri Lanka Institute of Information Technology the non-exclusive right to produce and distribute my dissertation in whole or part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as article or books).

Student Name	Student ID	Signature
Sandaruwan U.V.S.	IT21203244	
Dias A.H.J.S.S.	IT21203176	
Shamindi H.M.H.	IT21203558	
Priyawansha N.G.D.	IT21353284	

The above candidate is carrying out research for the undergraduate Dissertation under my supervision.

Signature of the supervisor Date

…… ……

(Mrs. Sanjeevi Chandrasiri)

ABSTRACT

The effective solution proposed by the researchers presented in this thesis paper aims to address the educational problems of children between the ages of 5 and 15 with Down Syndrome (DS), both domestically and internationally, and to overcome these problems and establish a successful foundation for their educational journey. The innovative approach presented utilizes machine learning, neural networks, image, and audio processing, and consists of four main components that encompass the VARK (Visual, Auditory, Reading, Writing, Kinesthetic) theory. The play activities have been developed covering the VARK theory, targeting the identifications made by children with DS, such as their interest in engaging in creative play-based learning activities and their preference for visual representations. This interactive, personalized adaptive e-learning platform ensures the scrutiny and engagement of every child, and tracking of progress ensures the most effective implementation of the child's educational journey.

By identifying DS, one of the most common chromosomal disorders worldwide, at an early stage and providing these children with proper guidance and guidance, the likelihood of them being labeled as children with special needs in society can be minimized. Although these children are provided with proper guidance in medical science to achieve their social background, due to factors such as the traditional education system and the low level of knowledge of parents and teachers, they are significantly lower in education compared to normal children at the initial stage. However, it has been found from the tests conducted with these children that when they are given personalized, properly guided education, they engage in it with great enthusiasm. Accordingly, the primary objective is to ensure the right of every child to receive an education by providing a personalized, adaptive e-learning platform that constantly monitors their progress and provides proper guidance.

Keywords: Down Syndrome, Neural Networks, Machine Learning, VARK Theory, Image Processing, Audio Processing

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to our project supervisor, Ms. Sanjeev Chandrasiri, for her continuous guidance and unwavering support, who played a crucial role in the successful completion of our undergraduate research. In addition, we express our heartfelt gratitude to our co-supervisor, Ms. Buddhima Attanayake, who was ready to assist us throughout the research project.

We express our deep gratitude to our Lecturer-in-Charge Dr. Jayantha Amaraarachchi and the CDAP team for their unwavering support in ensuring the success of the project. Furthermore, we are extremely grateful for the constant encouragement and support of our friends and team members.

We also express our sincere gratitude to Dr. Supun Premaratne, who served as our external supervisor. Mr. Premaratne's expertise enriched our research. We would like to express our gratitude to the Principal of Navodaya Special Children's Foundation, Moratuwa, Mrs. Lakshmi Karunajeewa, and the teaching staff for allowing us to collect data. We greatly appreciate their contribution to our research and their commitment to making this application a success in practical ways.

Finally, we would like to express our deepest gratitude to our parents for their continued support and encouragement throughout this journey.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1 INTRODUCTION

1.1 Background Literature

Down syndrome (DS), a genetic syndrome caused by an extra chromosome 21 [1], is one of the most common chromosomal disorders worldwide, with an estimated incidence of 1 in 1,000 to 1 in 1,100 live births. The pathology causes a range of cognitive problems, including difficulties with language, visual perception, and intellectual abilities, and directly impacts children's development, causing significant delays in reaching developmental milestones within the appropriate age range. Each child with DS has unique physical, behavioral, functional, and cognitive abilities. The prevalence of this genetic condition in Sri Lanka is estimated to be approximately 1 in 700 live births [2], which is substantially in line with global averages, making DS one of the most common chromosomal abnormalities in the country.

The intellectual levels of these children vary from one to another. It is classified into mild Intelligence Quotient (IQ) [50–70], moderate (IQ 35–50) and severe (IQ 20–35) [3] according to the level of cognitive impairment. According to the data, children with mosaic [4] DS syndrome have an IQ between 10 and 30. The majority of the above groups show moderate intellectual disabilities. The changes are evident in children in early childhood and these changes affect their visual-spatial and verbal memory, short-term memory, speech, language skills [5] [6] and social skills to varying degrees. It is a striking observation that children with DS use pictures, visuals [7], when they need to store something in their memory. Therefore, it has been confirmed that the use of visualization in the learning and teaching process for these children is a more effective method.

According to the factors highlighted above, children with DS face many difficulties and challenges in engaging in educational activities within the traditional educational framework [8]. They need specific and effective interventions with a good foundation at the early stage to develop their cognitive skills. Since the literacy level among parents and teachers about these diseases is low [2] in developing countries like Sri Lanka, these children miss out on factors that contribute to their development at an

early stage. It is essential to conduct further research on the learning styles, interests and dislikes of students with DS and to design learning approaches that enhance their abilities and contribute to their educational and social development. Here, the most important point that has been highlighted by experts and specialists in the field regarding these children with DS is that a personalized teaching and learning environment is more effective for these children [9]. The primary objective of this research is to develop a new adaptive learning methodological approach to identify the learning styles of DS students, monitor progress, visualize progress, and build a personalized learning environment using computer games and machine learning techniques. This research is built on the VARK (Visual, Auditory, Reading/Writing, Kinematic) [10] learning theory. The aim of the research is to create a personalized learning model that integrates language, education, and other skills with these VARK components by analyzing how teaching methods apply to children with DS. This personalized, adaptive e-learning platform is designed with a highly engaging user interface, color combinations, and a game environment specifically for students, with the aim of keeping students focused and engaged throughout the learning and teaching process. The platform has been designed with the insights and experience of education experts, healthcare professionals, and educators who regularly work with children with DS, contributing to everything from color combinations to activity creation. Machine learning techniques, user interaction analysis, and continuous training dataset utilization, create machine learning models that can identify students' learning challenges and learning preferences, and continuously evaluate and improve learning outcomes, ensuring the effectiveness of the system.

The primary objective of the proposed system is to improve the visual, auditory, motor, and reading/writing skills of children with DS in a balanced manner and guide students with DS to reach their full potential by working together with their peers in society, helping to minimize the difficulties they face. Through data-driven insights and adaptive learning techniques, this research aims to advance the field of special education, ultimately advocating for the right of every child to receive an education tailored to their individual needs.

1.2 Research Gap

A research need exists to develop an integrated multimodal educational system which unites auditory and visual elements with manual learning techniques and linguistic training approaches for children with Down Syndrome (DS). Studies about sensory aid solutions [11] examine individual modalities isolated from one another without assessing their interconnected effects on the cognitive development of children between ages 7 to 11 who have DS. The lack of an adaptively linked learning system [12] hinders the effective results of present-day solutions.

Auditory learning tools [13] currently offer only passive methods such as audiobooks or static speech tests without adaptive challenge adjustment or real-time assessment functions or gaming components. Current developments in technology through Google Speech-to-Text and DeepSpeech lack implementation within adaptable educational platforms designed for DS children who need support in memory development together with student focus and decision-making ability [14]. Research regarding real-time auditory content adaptation through reinforcement learning continues to be an uncharted area.

Declarative word knowledge and automated learning systems [15] employ some customization abilities through fixed educational materials and straightforward evaluation through test results. Learner profiles become limited in their adaptability because the systems do not deeply analyze cognitive-behavioral data including attention span, learning pace and error patterns. Research today fails to combine advanced NLP or ML techniques with learning pathway refinement and does not use effective gamification methods to enhance language learning in users who possess different developmental characteristics.

The learning technique known as kinesthesia faces specific difficulties when applied to reading and writing activities. The competencies demonstrated by deep learning models for recognizing DS handwriting features through studies appear limited because they need real-time guidance alongside adaptive features and accessible visualization support mechanisms. The development of practical accessible

handwriting solutions becomes challenged as we face two major limitations: the restricted availability of proper datasets and the existence of offline models coupled with the shortage of automated scaling feedback systems. Studies have neglected to address the physical issues of hypotonia that handicapped mouse or pen input [16] for children with DS while demonstrating a need for touch-friendly or gesture-based interfaces that support their needs.

The current offerings for visual and cognitive enhancement neglect [17] individual needs and do not provide proper assistance for children through personalized visual exercise development. A widespread design problem exists in technological systems because they implement standard solutions which disregard the varying developmental patterns in Down syndrome patients. These tools generally operate without structured feedback mechanisms [18] and the necessary reinforcement features or adaptive visual aid systems which match the learning stage of everyone. Research acknowledges the value of visual-cognitive skills that include facial recognition and color differentiation, and spatial understanding and short-term memory retention [19] yet fails to present a single solution that meets individual requirements by adapting to interactive visual tasks.

System adaptability lacks the integration of educational and parent feedback in all existing domains. Modern educational programs record exploratory information but most of them fail to use analytical systems for external input integration during real-time adaptive learning delivery [20] [21]. The majority of these tools lack predictive features to estimate time needed for mastery or identify trouble spots that would help spot progression and design intervention strategies.

An interactive ML-based multimodal learning platform under development as Blooming Minds specifically targets Down syndrome children between ages 5 to 15 to overcome critical knowledge deficits in education. The system includes auditory learning through speech games as well as adaptive NLP and ML-controlled verb learning modules together with kinesthetic exercises that have CNN-based evaluation followed by personalized visual-cognitive games which operate in real-time. The technology integrates three core elements including parent/educator feedback

systems, reinforcement learning models as well as progress prediction platforms to create a completely customized education environment. This solution combines all learning methods to manage the cognitive, motor, auditory and visual educational needs of DS students thus creating enhanced adaptive evidence-based educational technologies for the future.

Table 1.1: Research Gap based on previous research vs proposed solution

Feature	Blooming Minds	Gimpanzees	DSE – See and Learn	Magrid	Writing Wizard	Dynseo COCO
English Language Support	✓	✓	✓	✓	✓	✓
Math Hand Features	✓	✗	✗	✗	✗	✗
Real Time Audio Capturing	✓	✗	✗	✗	✗	✓
Real Time Motion Capturing	✓	✗	✗	✗	✗	✗
Letter Recognition and Accuracy Analysis	✓	✗	✗	✗	✓	✗
Color Sequence Memorizing Game	✓	✗	✗	✗	✗	✗
Progress Tracking	✓	✓	✓	✓	✓	✓
Future Learning Predictions	✓	✗	✗	✗	✗	✗

1.3 Research Problem

The educational path and academic performance of children with Down syndrome becomes seriously obstructed because of multiple neurological developmental limitations together with sensory differences. A genetic abnormality with chromosome 21 excess leads to DS which produces intellectual difficulties while triggering diminished cognitive speed and delayed speech development and problems

with memory and sensory coordination and attention span. The educational practices based on passive learning and one-size-fits-all teaching methods using verbal methods prove ineffective for meeting today's varying student requirements. The learning progression of children with Down syndrome becomes difficult due to deficits in basic cognitive competencies that include visual processing together with auditory comprehension and reading and writing abilities and language skills needed for academic growth and independent functioning.

The major cognitive difficulties faced by individuals with Down syndrome involve short-term memory limitations together with visual and auditory processing issues and abstract thinking deficits that prevent their ability to understand verbal instructions and process information while generalizing what they learn. Down syndrome children face difficulties persisting in classroom activities because current educational resources do not offer customization features or instant tailored feedback thus impeding their learning of communication abilities and literacy development. Studies about developing crucial skills like reading and writing along with language comprehension remain limited because research is weak in offering specialized interventions for different developmental profiles and cognitive abilities.

This research develops a complete learning enhancement program with four essential parts named Visual Learning Enhancement and Reading and Writing Development and Auditory Learning Support and a Kinesthetic Game-Based Language System (VerbMaster). The proposed system provides a personalized learning environment for children with Down syndrome through its combination of interactive digital tools with adaptive learning technologies and real-time feedback along with game-based and

engaging environments. The study investigates how the integrated tools assist memory storage through sight and sound and enable language mastery while establishing reading competence and boosting participation by using physical movements.

Ultimately, the study seeks to contribute to the creation of dynamic, accessible, and adaptive educational solutions that empower children with Down syndrome to thrive both academically and socially by unlocking their full potential.

1.4 Research Objective

Main Objective

The intention is to ensure that kids with Down syndrome receive equitable educational support via focused visible and cognitive enhancement. Cognitive and short-term memory improvement are foundational to capabilities such as pattern popularity, sequencing, reminiscence retention, and hassle-solving. This application pursuits to enhance those areas through integrating dependent visual sports, interactive gear, and adaptive strategies that align with every learner's unique cognitive profile. The number one objective is to foster educational increase and engagement via improving visual processing and cognitive characteristics in inclusive learning surroundings

- Visual Enhancement System (Component 1)
- Read and Write Enhancement System (Component 2)
- Auditory Enhancement System (Component 3)
- Kinesthetic Enhancement System (Component 4)

Each component has a specific objective and contributes to addressing the overarching research problem. The entire system will be implemented as a user-friendly web application.

Specific Objectives

Visual Enhancement System:

- **Sub Objective 1:** Designing and growing a coloration matching collection game to improve visual, cognitive abilities and to decorate brief term memory.
- **Sub Objective 2:** Designing and creating finger counting game to improve cognitive abilities.
- **Sub Objective 3:** Designing and creating basic math concept game: math marvels to improve visual and cognitive abilities

Read and Write Enhancement System:

- **Sub-Objective 01:** Developing a responsive web application which includes the letter writing activity
- **Sub-Objective 02:** Implementation of reading game.

Auditory Enhancement System:

- **Sub-Objective 01:** Design and develop engaging and interactive audiobook games.
- **Sub-Objective 02:** Create and integrate scoring mechanisms to evaluate user performance within the audiobook games**.**

Kinesthetic Enhancement System:

- **Sub-Objective 01:** Develop an innovative and interactive game designed to significantly enhance the recognition, understanding, and application of verbs within the context of language learning.
- **Sub-Objective 02:** Enhance Memory Retention by Associating Words with Physical Actions

Recommendation and Analysis using Rainforce learning.

The above 4 sub-objectives contribute towards completing the entire project functionality. The objectives and their development, implementation and results will be further discussed in the methodology section when the project functionality is explained and illustrated**.**

2 METHODOLOGY

This section comprises three sub-sections that explain the methodology of the entire system. These sub-sections discuss and illustrate the approach, the implementation and the testing of the product and the commercialization approach of the product.

2.1 Methodology

Data Collection

Requirement elicitation is a crucial step in system development. It is a phase where information is collected and analyzed from the users to specify both the functional and non-functional system requirements. In undertaking the prerequisite collection, the areas of research involved, and background studies were reviewed. A review of existing processes and comparable systems was also done following the analysis of the criteria. The project scope is already determined, and the system domain coverage can be considered a task. The main aim of this study is to collect hand-written sample data of letters, audio related data, face expressions and test data from DS children in the age group of 5-15.

Visual enhancement-based cognitive learning component

A specialized dataset was created for the visual enhancement of learning model through purposely designed interactive gameplay which targeted educational needs of Down syndrome children. Users interacted with memory tasks that involved color sequences along with image matching games that required dragging pictures which were accessible through the front-end application. User data was acquired in real-time during gameplay activities which contained variables measuring performance accuracy alongside completion time and number of attempts as well as sequence recognition patterns.

The research team collected the data first-hand from user sessions to achieve both relevance and proper match for children ages 7–11. Anonymization applied to all acquired data to protect user privacy. Specialists in child development teamed up with special education experts to determine success indicators together with ideal difficulty settings and suitable mental tasks for children at this age.

Read and write based cognitive enhancement component

The proposed application is developed using an online learning approach. Therefore, the data required to train the model for the reading development activity (user performance data) will be involved in the training process while the system is in use. Therefore, no previous data set was used to train the machine learning algorithm.

The children's confidentiality was upheld while gathering the data. In all, requirements gathering is a significantly critical stage in creating the suggested system. It consists of accumulating and analyzing data from users to confirm that the system aligns with their needs and expectations, and ultimately, to ensure user satisfaction and system efficiency.

Audio Enhancement based cognitive enhancement system component

To understand how children with Down syndrome respond to auditory-based learning, a survey was conducted at Navodaya Primary School. Questionnaires were distributed to teachers and caregivers to gather insights into the children's auditory preferences, learning challenges, and attention span during audio-based activities.

Dr. Supun Premaratne, our external supervisor with expertise in special education, provided valuable information related to auditory learning techniques, speech development in children with Down syndrome, and suitable learning methods for this user group.

Kinesthetic Enhancement system component

The data collection process for the VerbMaster project involves gathering relevant educational content and user feedback necessary to develop an engaging and effective learning tool. The following data sources were identified and collected to support the development and testing of the application. Data was collected from potential users, including students and teachers, to better understand user needs and expectations. This data was analyzed to inform user interfaces and the development of game mechanics that align with educational goals.

System Architecture and Approach

The depiction below represents the base structure which functions as the fundamental design of the cognitive enhancement system built for Down syndrome children. The main objective behind this system creates a multi-modal learning solution designed to support learners with Down syndrome through their developmental and cognitive and sensory needs.

These specialized modules in the architecture system present Visual and Read-Write together with Auditory and Kinesthetic enhancement activities which focus on different cognitive areas for development. Through a user interface designed for children a system allows collaborative access for teachers alongside caregivers and parents. Through the interface users access all four modules to facilitate supervised activities under customized choices with direct monitoring of current levels of achievement.

The Overall System flow is as follows:

The Visual Enhancement Module through drag-and-drop games combined with visual sorting tasks and pattern recognition activities followed by sequencing exercises. The visual processing accuracy and response time as well as recognition skills of users are assessed through analysis of Convolutional Neural Networks (CNNs). The Read-Write Module addresses letter development by teaching formation along with tracing skills along with keyboard typewriting abilities. The system analyzes accuracy in spelling and writing patterns and typing performance through CNN and Support Vector Machine (SVM) model processing. The detection system employs these models to identify writing improvements with its additional ability to find frequent errors alongside measuring input latency.

The Auditory Enhancement Module executes functions for both verbal instructions and letter-word identification and sound repetition. User responses together with auditory memory performance get examined by CNN models specialized in speech detection and audio pattern recognition.

The VerbMaster component uses motion-based input as it enables users to engage in game-based activities that match images or actions by physical gestures. The system implements gamified learning loops to enhance both memory skills and vocabulary development in addition to promoting motor coordination functions.

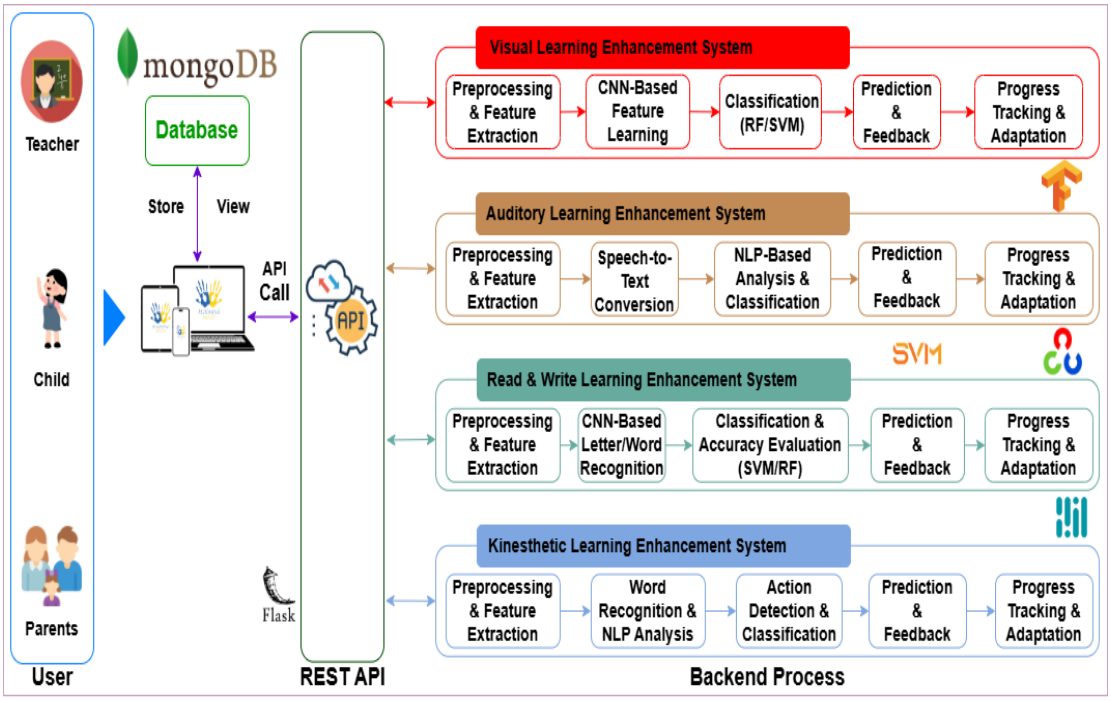

Figure 2.1: Complete system overview diagram

Component system architecture diagrams

1. Visual Learning Enhancement System

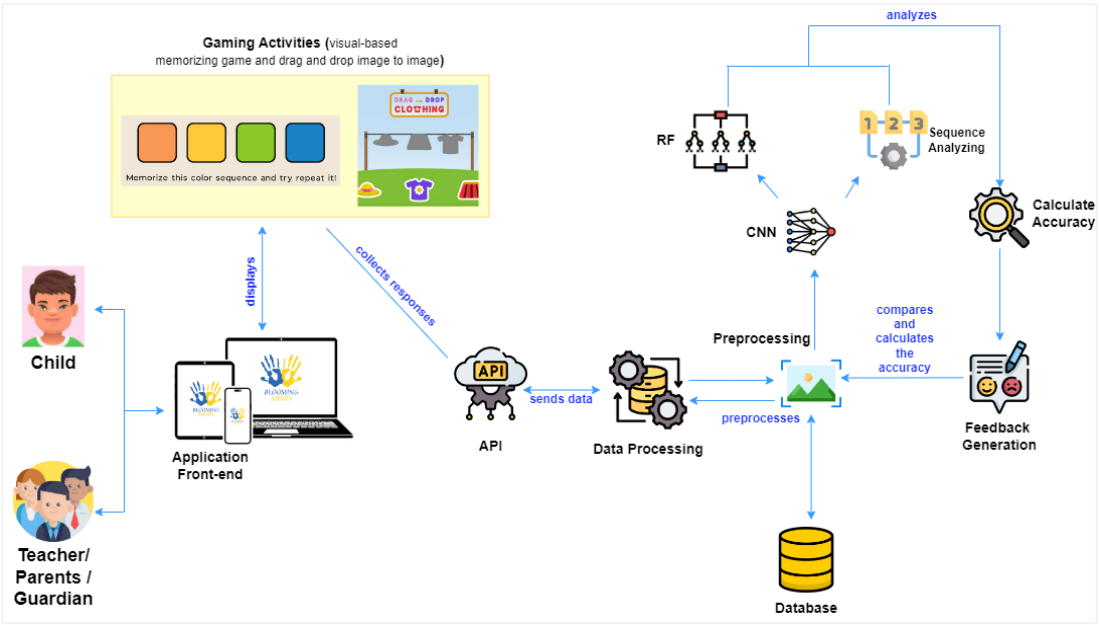

Figure 2.2: Visual learning enhancement system overview diagram

2. Read and Write Learning Enhancement System

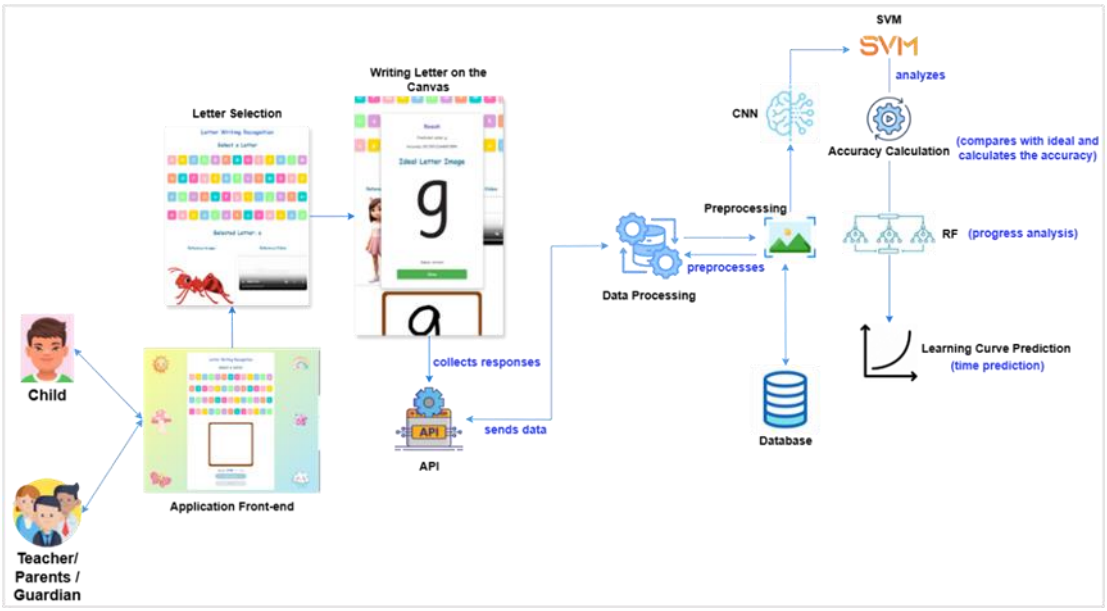

Figure 2.3: Reading game system overview diagram

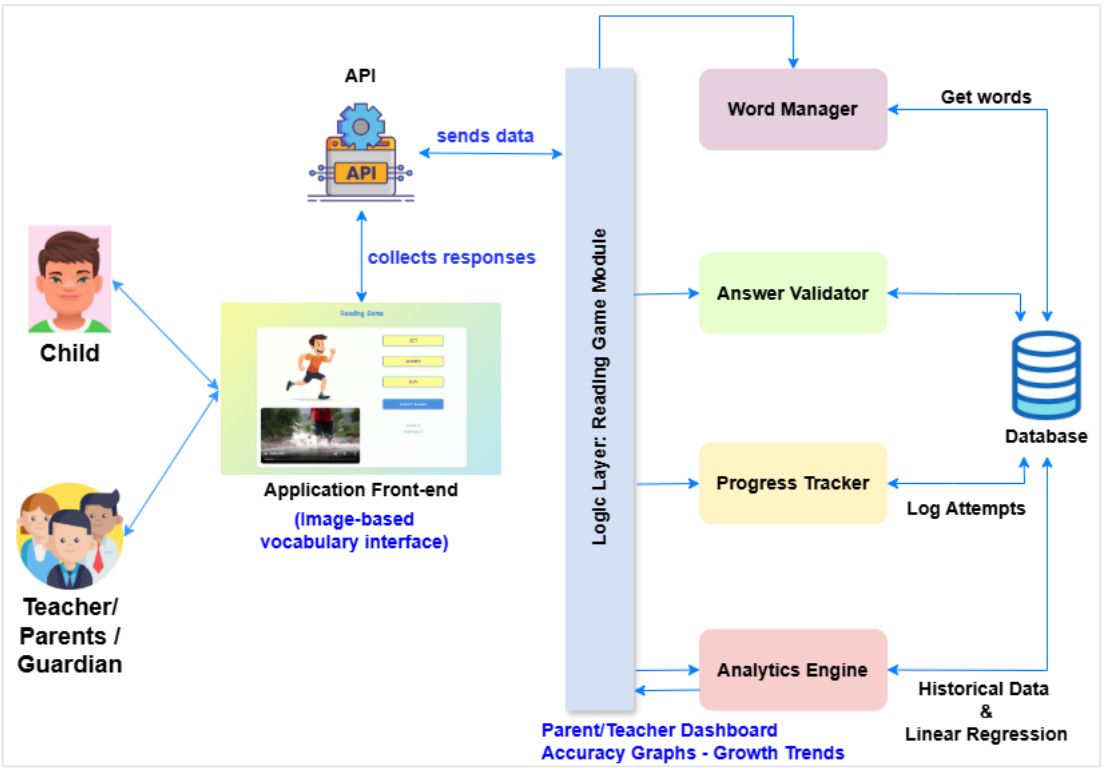

Figure 2.4: Reading game system overview diagram

3. Auditory Learning Enhancement System

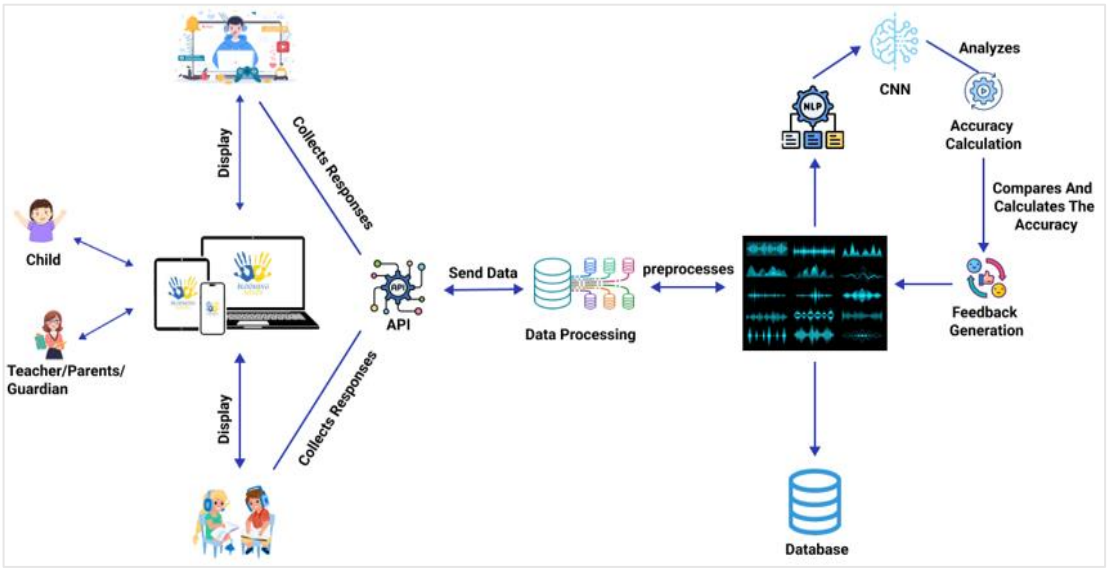

Figure 2.5: Auditory learning enhancement system overview diagram

4. Kinesthetic Learning Enhancement System

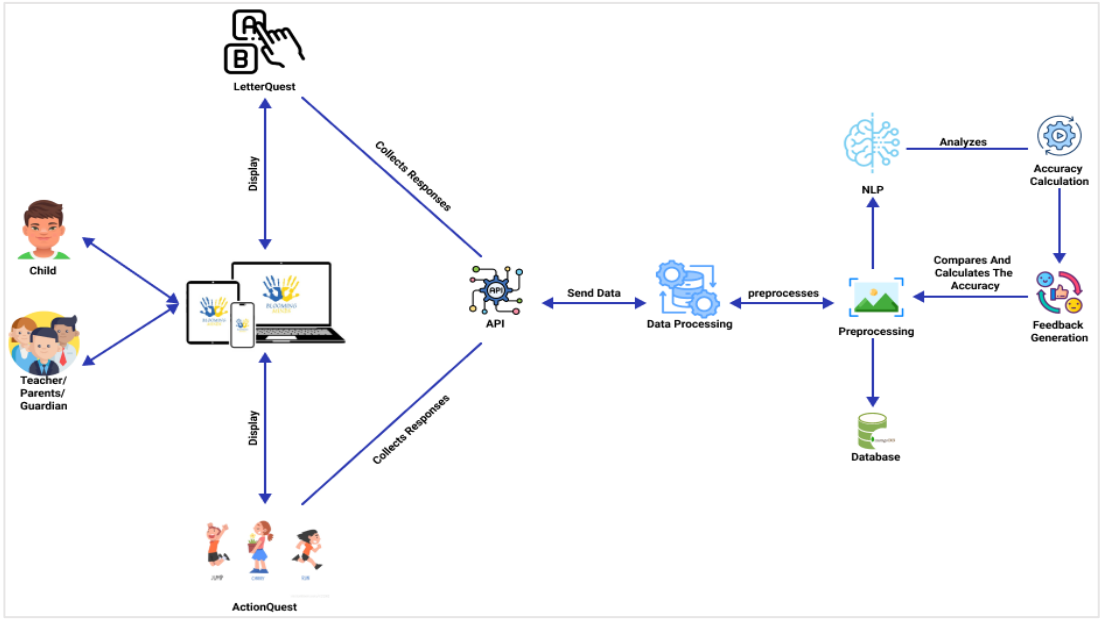

Figure 2.6: Kinesthetic learning enhancement system overview diagram

Tools and Technologies

Table 2.1: Tools and technologies

Description	Tools and Technologies
Programming IDE	Visual Studio Code, Jupyter Notebook
Programming language for responsive web application development	ReactJS
Machine learning algorithm-based predictions and analysis	Python Language, TensorFlow (CNN/ML models) , Keras, Natural Language Toolkit (NLTK)
Programming language for backend development	Flask, Speech Recognition Library
Database for store data	MongoDB
Hosting the API	Flask, Heroku
Version Controlling	Github
Team connectivity	MS Teams and WhatsApp

The table presented above (Table) offers a comprehensive account of the tools and technologies employed in the development of the application as well as the program.

2.2 Commercialization Aspects of the Application

The commercialization of Blooming Minds - an adaptive e-learning platform for developing the learning and teaching process of children with Down syndrome focuses on two subscription plans: a freemium version for individual users and a premium version for organizations, institutions, or special education units. The platform is designed to provide tailored educational experiences for children with DS, providing individual and organizational value, making it a highly scalable and adaptable solution in the special education market.

2.2.1 Target Market

- Individual users: Parents and guardians of children with DS looking for an educational tool to enhance their child's learning process.
- Organizations: Special education institutions, school clinical centers, and non-profit organizations that work with children with DS and other learning disabilities.
- Special education sector: Public or private sector organizations that aim to support individuals with cognitive or learning disabilities.

As the platform grows, additional markets may be targeted, including early childhood education centers and other organizations interested in inclusive education technology.

2.2.2 Marketing and Revenue

Figure 2.7: Pricing strategy

Table 2.2: Pricing plan

Freemium	Premium
$0.00 per month	$4.99 per month
Color Matching Game	Math Marvels Game
Image-to-Word Mapping Game	Letter Writing Game
Audio Book Game	Finger Counting Game
Letter Quest	Action Quest
Real Time Feedback	Recommendation and Prediction

Here above the Table 2.2 and Figure 2.7 depicts the pricing plan of 'Blooming Minds' that follows as marketing strategy. Our team considered the following options as a revenue generation model.

- Premium version
- Donations, fundings and sponsorships
- Projects
- Workshops and trainings
- Advertisements
- One-Time payment plan

Phase 01: Testing and Feedback

– Initializing a pilot version of software to a specific user (children, teachers and parents) to gather feedback based on real world, this phase is used to refine the platform using the insights gathered during the pilot session.

Phase 02: Freemium and Premium Versions

- Launched with the aim to launch a paid subscription based professional version of the platform for which individual users can enjoy and schools and organizations can enjoy.

Phase 03: Targeted Campaigns

- Social media platform, specialized forum and partnership with DS organizations implement the targeted marketing campaigns. Therefore, the focus should be on the demonstrability of improving the effectiveness of the learning outcomes of these children by using the platform.

Phase 04: Continuous Improvement

- Collect ongoing feedback on details of the platform from individual users and organizations while constantly updating the platform. Provide user demand based new features and improving the existing features for achieving high customer satisfaction. Write positive testimonials and case studies.

Phase 05: Partnerships and Expanding Reach

- Partner with special education advocate groups, government agencies as well as with technology giants that share the same intend of inclusive education and consider the possibilities to forge a partnership with educational institutions and technology suppliers in order to boost the availability of Blooming Minds throughout the globe.

Therefore, well focused on the underserved market of people with Down syndrome and their caregivers, Blooming Minds has strong business potential.

2.3 Implementation

The cognitive enhancement platform organizes its functionalities into four distinct elements which we name Visual, Read-Write, Auditory, and Kinesthetic. The program components function autonomously to capture real-time data from users, apply machine learning to test performance, and create feedback. The following section explains how to set up each module in the platform of content enhancement.

Visual Enhancement Module Implementation

Accordingly, before probing into enforcing the proposed result, needed to determine precisely what the user concerns from the system, what the system needed to perform, and what supplementary conditions it required to negotiate. The associated concerns are as follows.

User Requirements

- The system must support personalized learning approaches through customization to individual child cognitive development progress.

- Children should be able to engage with the system because it features an accessible interface with easy-to-use methods. Children with DS need interfaces that have easy navigation and understanding capabilities through implementation of strategies that include large icons and clear directions.

- Parental and educator monitoring should remain easy while receiving performance data that enables clear recommendation creation for further child development.

Users require flexible access since the system enables usage of tablets, smartphones and computers. This component matches user’s needs.

Functional Requirements

- Scalable learning algorithms in the system would automatically modify visualization memory tasks difficulty levels according to each child's results.
- The system features a Finger Counting Game Interface for improving children's mental processes as well as mental awareness skills and their response times.
- The system must recognize and verify color sequences in real time with Convolutional Neural Networks (CNN) for providing accurate feedback.
- Through its guiding function the system will supply quick feedback to users and automatically modify the difficulty settings of activities through performance-based adaptation.
- The gathered data must be used to train models by assessing performance data which comprises accuracy levels and speed metrics alongside error rate measurements.

Non-Functional Requirements

- The system should process information in real time to deliver immediate feedback throughout a continuous learning session.
- The system needs to scale up its capacity to handle large user bases which proves essential for educational institutions together with therapy centers.
- The system requires encryption along with secure storage of user data specifically child-related data to meet all privacy regulations.
- The platform needs to have an interface adjusted to serve the special requirements of Down syndrome children together with their support network while also being easy to navigate for users.

- Learning sessions must run without interruption because the system needs to be reliable through its continuous availability and limited time delays.

System Requirements

- Hardware
  - Tablets, smartphones, or computers with touch interfaces (for handwriting input).
  - Stable internet connection for accessing cloud-based features.
- **Software**
  - Cloud Infrastructure: The system should use a cloud-hosted database to store user
  - data, game progress, and analytics.
  - Machine Learning Models: Integration of CNNs and adaptive learning algorithms for
  - real-time input analysis and feedback.
  - Secure APIs: Secure communication between the front-end user interface and the
  - back-end data processing services.

After considering the above-mentioned requirements had been considered, the team decided to build a responsive web application that employs innovative methods to overcome the limitations and challenges faced by children with DS during their education journey. I divided the load and kickstarted the development process into three key areas**.**

1. Implementation of 'Color Matching Sequence Game'
2. Implementation of 'Finger Counting Game'
3. Implementation of 'Math Learning Game with Number Cards'

Implementation of 'Color Matching Sequence Game'

The 'Color Matching Sequence Game' is one of the core components of the Visual Learning Enhancement System, designed to assess and support the development of visual and short-term memory. This module evaluates the ability to remember, chromatic vision ability and based on the user performance the system assesses and recommend the difficulty level.

This system represents a sophisticated perpetration of adaptive literacy technology combined with underpinning literacy and prophetic modeling. This comprehensive system was designed to give individualized literacy gests for children by stoutly conforming difficulty situations, offering targeted color suggestions, and satisfying progress. The perpetration integrates multiple factors including a Beaker- grounded web service, MongoDB for data continuity, underpinning learning for adaptive difficulty, and LSTM networks for color sequence vaticination.

1. **System Architecture and Core Components**

The backend system follows a modular armature with clear separation of enterprises between different functional areas. The main entry point is the Flask operation defined in app.py, which sets up the web server and registers colorful route arrangements. The color matching game functionality is reprised in the color_matching_routes.py module, which handles all game related HTTP endpoints. This modular approach allows for clean association of law and makes the system more justifiable and extensible.

Data continuity is managed through MongoDB, with database operations abstracted in the db_util.py module. This setup provides inflexibility in data storehouse and allows for effective querying of game results and user biographies. The system maintains several collections in MongoDB including color_matching_results for storing game session data, user_color_profiles for tracking individual stoner patterns, and prices for managing the price system.

The machine literacy factors correspond of two main corridor the RLAgent for underpinning literacy- grounded difficulty adaptation and the ColorPredictor for generating substantiated color suggestions. These factors work together to produce an adaptive literacy experience that responds to the stoner's performance in real- time.

2. Game Session Processing and Performance Analysis

The core of the adaptive difficulty system is the RLAgent class, which implements a Deep Q-Network (DQN) reinforcement learning algorithm. This agent learns to recommend appropriate difficulty levels based on the user's performance:

```
14  class RLAgent:
15      def __init__(self, state_size, action_size, model_path=None):
16          self.state_size = state_size
17          self.action_size = action_size
18          self.memory = deque(maxlen=2000)
19          self.gamma = 0.95  # Discount factor
20          self.epsilon = 1.0  # Exploration rate
21          self.epsilon_decay = 0.995
22          self.epsilon_min = 0.01
23          self.learning_rate = 0.001
24          self.model = self._build_model()
25          self.total_rewards = []  # Track total rewards per episode
26          self.epsilon_values = []  # Track epsilon values per episode
27          self.loss_values = []  # Track loss values during training
28          if model_path:
29              self.model.load_weights(model_path)
30
```

Figure 2.8: DQN-RL algorithm

The agent uses an ε-greedy policy for disquisition vs. exploitation, starting with high disquisition(ε = 1.0) and gradationally decaying this value to favor exploitation of learned knowledge. The neural network model consists of two hidden layers with ReLU activation and a linear output layer representing Q- values for each possible action(difficulty level).

- **ε-Greedy Policy for Exploration vs. Exploitation**

The agent uses the ε-greedy policy to balance disquisition(trying new action to discover their effects) and exploitation(choosing the best- known action grounded on current knowledge).

- ε (epsilon) A value between 0 and 1 that determines the probability of taking an arbitrary action (exploration) vs. the best-known action (exploitation).
    - When ε = 1.0, the agent always explores (100% random actions).
    - When ε = 0.0, the agent always exploits (always picks the action with the highest predicted Q-value).
- Decay Over Time
    - The agent starts with high disquisition(ε = 1.0) to gather different experiences.
    - When ε = 0.0, the agent always exploits (always picks the action with the high est predicted Q-value).

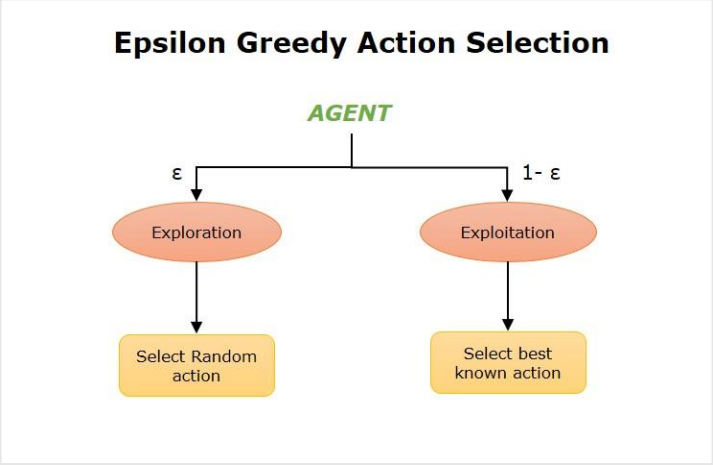

Figure 2.9: Epsilon greedy action selection diagram

- **Neural Network Model Architecture**

The agent uses a neural network to compare the Q- function, which estimates the anticipated future prices for taking each possible action(difficulty level) in each state.

- Input Layer
  - Represents the current state of the terrain(e.g., features like the player's skill position, performance hsitory, etc.).
- Hidden Layers(2 layers with ReLU activation)
  - First Hidden Layer: Takes the input and applies weights and impulses, followed by a ReLU(remedied Linear Unit) activation function

$$\mathbf{ReLU}(x)=\mathbf{max}(0,x)$$

  - ReLU introducesnon-linearity, helping the network learn complex patterns.
- Second Hidden Layer: Another ReLU-activated layer to further process the data.
  - Output Layer (Linear): Produces the estimated Q-values for each possible action (difficulty level).

The state representation is crucial for effective learning. The state_from_performance function transforms raw performance metrics into a normalized state vector:

```python
def state_from_performance(performance):
    """
    Generate state vector from performance metrics.
    """
    level = LEVELS.get(performance["level"], 0)
    wrong_takes = performance["wrong_takes"]
    correct_takes = performance["correct_takes"]
    time_taken = performance["time_taken"]

    # Calculate additional metrics
    total_attempts = correct_takes + wrong_takes
    accuracy_percentage = (correct_takes / total_attempts) * 100 if total_attempts > 0 else 0
    average_time = 30  # Assume an average benchmark time (can be dynamic)
    time_efficiency = average_time / time_taken if time_taken > 0 else 1

    # Normalize values and construct state vector
    return np.array([level, wrong_takes, correct_takes, time_taken, accuracy_percentage, time_efficiency]).reshape(1, -1)
```

Figure 2.10: state_performance code snippet

During game processing, the agent:

- Receives the current state deduced from performance criteria
- Selects an action(recommended difficulty position)
- Receives a price grounded on the quality of the recommendation
- Stores this experience in its renewal memory
- Periodically trains on batches of once gests

```python
minibatch = random.sample(self.memory, batch_size)
loss = 0
for state, action, reward, next_state, done in minibatch:
    state = np.expand_dims(state, axis=0)
    next_state = np.expand_dims(next_state, axis=0)
    target = reward

    if not done:
        target = reward + self.gamma * np.amax(self.model.predict(next_state)[0])
    target_f = self.model.predict(state)
    target_f[0][action] = target
    history = self.model.fit(state, target_f, epochs=1, verbose=0)
    loss += history.history['loss'][0]

self.loss_values.append(loss / batch_size)  # Average loss for the minibatch
```

Figure 2.11: State performance code snippet II

3. LSTM-Based Color Sequence Prediction

The ColorPredictor class implements an LSTM (Long Short-Term Memory) network to generate personalized color sequences based on the user's historical performance:

The ColorPredictor class uses an LSTM(Long Short- Term Memory) network, a technical type of intermittent neural network(RNN) designed to handle sequential data. Unlike standard feedforward neural networks, LSTMs have a "memory" that allows them to learn patterns across time way, making them ideal for tasks like making predictions about the color suggestions in a sequence. This is because colors in a sequence frequently depend on former colors(e.g.," red → blue → green" may follow a pattern that the LSTM can learn).

The ColorPredictor class compromise:

- Embedding Layer
  - This layer maps integers to dense vectors, when the colors are label-encoded
- LSTM Layer
  - Processes the sequence step-by-step, streamlining its hidden state to capture dependences .
- Dense (Output) Layer
  - Charts the LSTM's final hidden layer to a probability distribution over possible color suggestion

```python
class ColorPredictor:
    def __init__(self, model_path=None):
        self.model_path = model_path or 'color_lstm_model.keras'
        self.color_encoder = LabelEncoder()
        self.known_colors = ['red', 'blue', 'green', 'yellow', 'orange', 'purple', 'pink', 'brown']
        self.color_encoder.fit(self.known_colors)
        self.max_sequence_length = 10  # Maximum sequence length to consider
        self.num_colors = len(self.known_colors)

        # Initialize model
        if os.path.exists(self.model_path):
            self.model = load_model(self.model_path)
            logger.info("Loaded existing LSTM model")
        else:
            self.model = self._build_model()
            logger.info("Created new LSTM model")
```

Figure 2.12: ColorPrediction class implementation

```python
def _build_model(self):
    model = Sequential([
        LSTM(64, input_shape=(self.max_sequence_length, self.num_colors)),
        Dropout(0.2),
        Dense(64, activation='relu'),
        Dense(self.num_colors, activation='softmax')
    ])
    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model
```

Figure 2.13: Model implementation

The model processes sequences of colors represented as one-hot decoded vectors and predicts the most likely coming color in the sequence. This allows the system to induce sequences that follow patterns the stoner has preliminarily plodded with, furnishing targeted practice.

The color suggestion generation combines the LSTM predictions with the user's mistake patterns:

```python
class ColorPredictor:
    def generate_suggestions(self, user_id, level='easy'):
        """
        Generate personalized color suggestions based on user's mistake patterns
        """
        # Get user's color profile from database
        profile = self.db.user_color_profiles.find_one({"user_id": user_id})
        if not profile:
            # If no profile exists, return a default sequence
            count = LEVEL_CIRCLE_COUNTS.get(level, 3)
            return np.random.choice(self.known_colors, size=count, replace=True).tolist()

        # Get user's mistake patterns
        mistake_patterns = profile.get("mistake_patterns", {})
        difficult_colors = sorted(mistake_patterns.items(), key=lambda x: x[1], reverse=True)

        # Determine sequence length based on level
        seq_length = LEVEL_CIRCLE_COUNTS.get(level, 3)

        # Generate sequence with more frequent difficult colors
        sequence = []
        if difficult_colors:
            # Include top 2 difficult colors
            top_colors = [color for color, count in difficult_colors[:2]]
            remaining_slots = max(0, seq_length - len(top_colors))

            # Fill remaining slots with other colors
            other_colors = [color for color in self.known_colors if color not in top_colors]
            sequence = top_colors + np.random.choice(other_colors, size=remaining_slots, replace=True).tolist()
        else:
            # No difficult colors identified, return random sequence
            sequence = np.random.choice(self.known_colors, size=seq_length, replace=True).tolist()

        # Shuffle the sequence to avoid predictability
        np.random.shuffle(sequence)

        return sequence[:seq_length]  # Ensure correct length
```

Figure 2.14: Color suggestion based on mistaken patter (use of LSTM)

This approach ensures that the suggested sequences concentrate on colors the stoner finds grueling while maintaining variety through arbitrary selection of other colors. The sequences are scuffled to help predictable patterns that might lead to rote memorization rather than genuine literacy.

4. Machine Learning Model Training and Maintenance

The LSTM model for color sequence vaticination is designed to be trained incrementally as further user data becomes available. The training process involves,

- Collecting historical color sequences and the subsequent colors users selected
- Preprocessing these sequences into one-hot encoded matrices
- Splitting the data into training and validation sets
- Training the model with early stopping to prevent overfitting

```python
def train_model(self, sequences, next_colors, epochs=20, batch_size=32):
    """
    Train the LSTM model on historical color sequences and next colors
    """
    # Preprocess sequences
    X = np.array([self._sequence_to_matrix(self._pad_sequence(seq)) for seq in sequences])
    y = self.color_encoder.transform(next_colors)
    y = np.eye(self.num_colors)[y]  # One-hot encode

    # Split data
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)

    # Train model
    history = self.model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=epochs,
        batch_size=batch_size,
        verbose=1
    )

    # Save model
    self.model.save(self.model_path)
    logger.info(f"Model saved to {self.model_path}")

    return history
```

Figure 2.15: ML model training

The reinforcement learning agent also benefits from continuous learning. As users interact with the system, the agent accumulates more experiences in its replay memory, allowing it to refine its difficulty recommendations over time. The agent's performance can be monitored through several metrics:

- Total reward per episode (should generally increase over time)
- Epsilon value (should decay appropriately)
- Training loss (should decrease and stabilize)

5. Performance Considerations

Several design opinions were made to ensure the system performs well.

- Database queries are optimized to only retrieve necessary data
- The RL agent uses experience replay with a fixed-size memory buffer
- Model predictions are batched where possible

- Heavy computations (like model training) are performed asynchronously
- Frequently accessed data (like user profiles) could be cached
- The system also implements various metrics (accuracy, time efficiency) that could be used to monitor performance and identify areas for optimization.

Implementation of 'Finger Counting Game'

The Blooming Minds finger counting game incorporates advanced features that unite computer vision methods with Flask backend functionalities and MongoDB database storage. Users navigate through the application by performing hand gestures that involves finger counting as an easy-to-use interface for educational purposes. The project incorporates multiple technical fields such as real-time image processing alongside machine learning-based hand detection and creates RESTful API functions and implements database systems.

1. **Core Architecture and System Design**

Throughout the game operation the client-server model allows the frontend device to record webcam hand images which the backend process thereafter. The system uses Flask as its backend framework as it delivers essential endpoints that focus on hand recognition and finger count processing. MediaPipe functions as the core computer vision component through its open-source role in multimodal perception tasks for hand landmark detection.

The application initiates by client-side JavaScript proceeds webcam frames from users before encoding them as base64 data to send to the /count_fingers API endpoint. The backend handles the image transmission through its hand detection pipeline to compute visible finger numbers that it returns to the frontend system. The gameplay mechanics based on finger counting run in real time through this live interaction.

2. **MediaPipe Hand Landmark Detection**

The MediaPipe Hands solution is used as the core of the finger counting implementation, providing accurate hand tracking and 21-point landmark recognition. Specific parameters were employed to ensure stable system performance and accuracy to starts the MediaPipe Hands model.

```
16  hands = mp_hands.Hands(
17      static_image_mode=True,
18      max_num_hands=2,
19      min_detection_confidence=0.5,
20      min_tracking_confidence=0.5
21  )
```

Figure 2.16: Parameter usage

Since our implementation is single frame, the static_image_mode parameter is set to True, and this configuration allows the model to treat each image independently. This is more appropriate for our use case, where we receive discrete images from the front end. The max_num_hands parameter is set to 2 because we want to enable more complex counting scenarios and allow for simultaneous detection of both hands. The confidence limits (min_detection_confidence and min_tracking_confidence) are set to 0.5, which provides a balance between detection accuracy and performance. These values were determined through practical tests to maintain high detection accuracy without interference in the presence of hands of different sizes and lighting conditions.

Thereafter, we will need to initialize the mp.solutions.hands class and then set up the mp.solutions.hands.Hands() function with appropriate arguments and initialize mp.solutions.drawing_utils class that is required to visualize the detected landmarks. We will be working with images and videos as well, so we will have to set up the mp.solutions.hands.Hands() function two times.

Once with the argument static_image_mode set to True to use with images and the second time static_image_mode set to False to use with these speeds up the landmark's detection process, and the intuition behind this was explained in detail in the previous post.

3. Finger Counting Algorithm

MediaPipe exposes the logic required to count fingers to represent a complex geometric analysis of the hand joints. The algorithm clearly distinguishes between different finger types.

```
51  def count_fingers(image):
52      """Count the number of fingers up for each hand in the image using the new logic.
53      Args:
54          image: The image of the hands on which fingers counting is required.
55
56      Returns:
57          total_count: Total count of fingers that are up from both hands.
58      """
```

Figure 2.17: Finger counting algorithm code snippet I

```python
def count_fingers(image):
    """Count the number of fingers up for each hand in the image using the new logic.
    Args:
        image: The image of the hands on which fingers counting is required.

    Returns:
        total_count: Total count of fingers that are up from both hands.
    """
    try:
        # Detect hand landmarks
        _, results = detect_hands_landmarks(image)

        if not results.multi_hand_landmarks:
            return 0

        # Initialize count dictionary
        count = {'RIGHT': 0, 'LEFT': 0}

        # Store the indexes of the tips landmarks of each finger
        finger_tips_ids = [
            mp_hands.HandLandmark.INDEX_FINGER_TIP,
            mp_hands.HandLandmark.MIDDLE_FINGER_TIP,
            mp_hands.HandLandmark.RING_FINGER_TIP,
            mp_hands.HandLandmark.PINKY_TIP
        ]

        # Iterate over the found hands in the image
        for hand_index, hand_info in enumerate(results.multi_handedness):
            hand_label = hand_info.classification[0].label
            hand_landmarks = results.multi_hand_landmarks[hand_index]

            # Check fingers (index, middle, ring, pinky)
            for tip_index in finger_tips_ids:
                finger_name = tip_index.name.split("_")[0]

                # Check if finger is up by comparing y-coordinates of tip and pip landmarks
                if (hand_landmarks.landmark[tip_index].y < hand_landmarks.landmark[tip_index - 2].y):
                    count[hand_label.upper()] += 1

            # Check thumb (special case)
            thumb_tip_x = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].x
            thumb_mcp_x = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP - 2].x

            # Check if thumb is up based on hand label and x-coordinates
            if (hand_label == 'Right' and (thumb_tip_x < thumb_mcp_x)) or \
               (hand_label == 'Left' and (thumb_tip_x > thumb_mcp_x)):
                count[hand_label.upper()] += 1

        return sum(count.values())

    except Exception as e:
        logging.error(f"Error in count_fingers: {str(e)}")
        return 0
```

Figure 2.18: Finger counting algorithm code snippet II

Perform hand landmark detection

In the step, where create a function `detectHandsLandmarks()` that will take an image/frame as input and will perform the landmarks detection on the hands in the image/frame using the solution provided by MediaPipe and will get twenty-one (21)

3D landmarks for each hand in the image. The function will display or return the results depending upon the pass arguments.

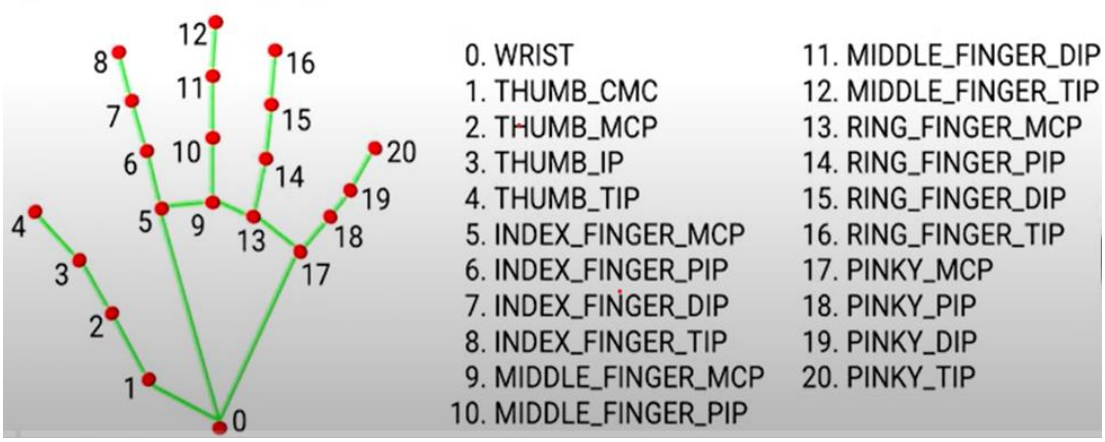

Figure 2.19: 21, 3D landmarks used in MediaPipe

All hand landmarks in the input image will identified by the algorithm as the initial step. It assesses four fingers (index, middle, ring, and pinky - for each detected hand) by comparing the vertical (y) position of their tips with their respective PIP (proximal interphalangeal) joints. If a fingertip is above (has a lower y-coordinate than) its PIP joint, the finger is considered extended and counted. Due to thumb's unique range of motion detection follows separate logic. The algorithm compares the horizontal (x) position of the thumb tip with its MCP (metacarpophalangeal) joint. Detection rule varies based on the hand is identified as left or right, accounting for the mirror-image nature of opposite hands.

4. Image Processing Pipeline

The image processing pipeline begins when a frame captured by the front end is sent to the back end as a base64-encoded string. This image is decoded and prepared for processing by the Flask route handler.

Now in this step, we will create a function `countFingers()` that will take in the results of the landmarks detection returned by the function `detectHandsLandmarks()` and will utilize the landmarks to count the number of fingers up of each hand in the image/frame and will return the count and the status of each finger in the image as well.

```python
def detect_hands_landmarks(image, draw=False):
    """Perform hands landmarks detection on an image.
    Args:
        image: The input image with prominent hand/s whose landmarks needs to be detected.
        draw: A boolean value that if set to true draws landmarks on the output image.

    Returns:
        output_image: A copy of input image with the detected hands landmarks drawn if specified.
        results: The output of the hands landmarks detection on the input image.
    """
    output_image = image.copy()
    imgRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = hands.process(imgRGB)

    if results.multi_hand_landmarks and draw:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                image=output_image,
                landmark_list=hand_landmarks,
                connections=mp_hands.HAND_CONNECTIONS,
                landmark_drawing_spec=mp_drawing.DrawingSpec(color=(255,255,255), thickness=2, circle_radius=2),
                connection_drawing_spec=mp_drawing.DrawingSpec(color=(0,255,0), thickness=2, circle_radius=2)
            )

    return output_image, results
```

Figure 2.20: Hand landmarks detection code snippet

```python
@hand_detection_routes.route('/count_fingers', methods=['POST'])
@token_required
def count_fingers_api():
    try:
        data = request.json
        image_data = data.get('image')

        if not image_data:
            return jsonify({"error": "Missing image data"}), 400

        # Decode base64 image
        _, encoded = image_data.split(",", 1)
        nparr = np.frombuffer(base64.b64decode(encoded), np.uint8)
        image = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

        # Count fingers using the new logic
        finger_count = count_fingers(image)

        return jsonify({
            "finger_count": finger_count
        })
```

Figure 2.21: Finger counting logic

The decoding process involves several steps:

- The base64 string is split to remove the data URL prefix
- The remaining string is decoded into a byte array
- OpenCV's `imdecode` function converts the byte array into a standard image matrix
- The image is then passed to the finger counting function

Verify image quality for accurate manual identification of the pipeline, ensuring compatibility with web-based image transmission.

- Operating Mechanism

To check the status of each finger (i.e., either it is up or not), we will compare the y-coordinates of the `FINGER_TIP` landmark and `FINGER_PIP` landmarks for each finger. Whenever the finger will be up, the y-coordinate of the `FINGER_TIP` landmark will have a lower value than the `FINGER_PIP` landmark.

But for the thumbs, the scenario will be a little different as we will have to compare the x-coordinates of the `THUMB_TIP` landmark and `THUMB_MCP` landmark, and the condition will vary depending upon whether the hand is left or right.

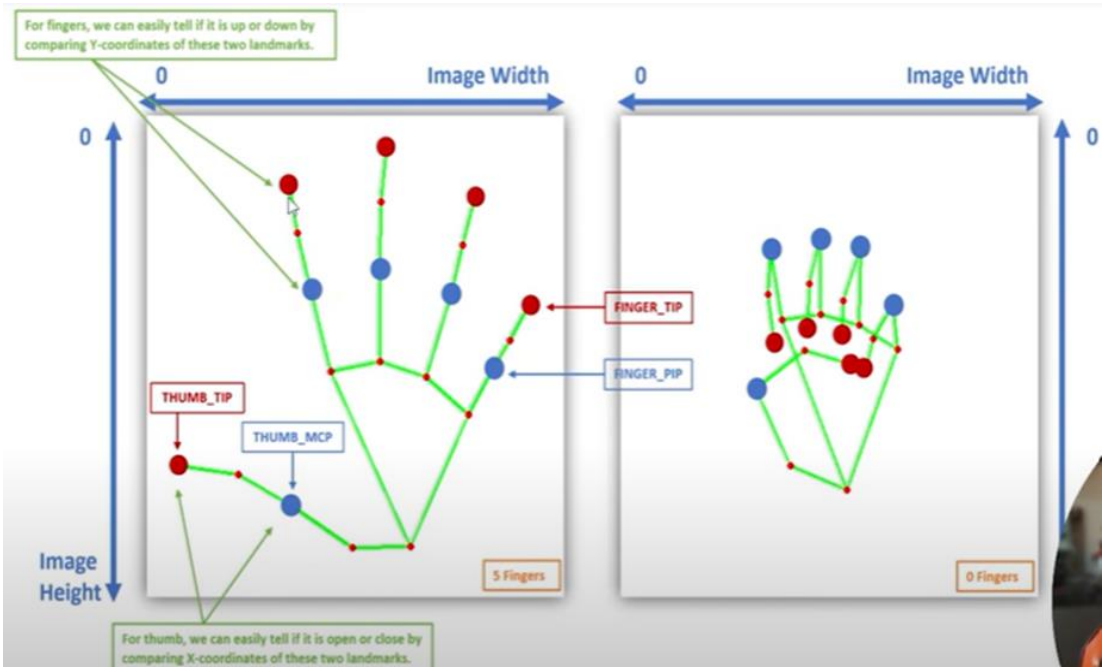

Figure 2.22: Tip based finger coordination

For the right hand, whenever the thumb will be open, the x-coordinate of the `THUMB_TIP` landmark will have a lower value than the `THUMB_MCP` landmark and for the left hand, the x-coordinate of the `THUMB_TIP` landmark will have a greater value than the `THUMB_MCP` landmark. Fig. 2.23 depicts this information.

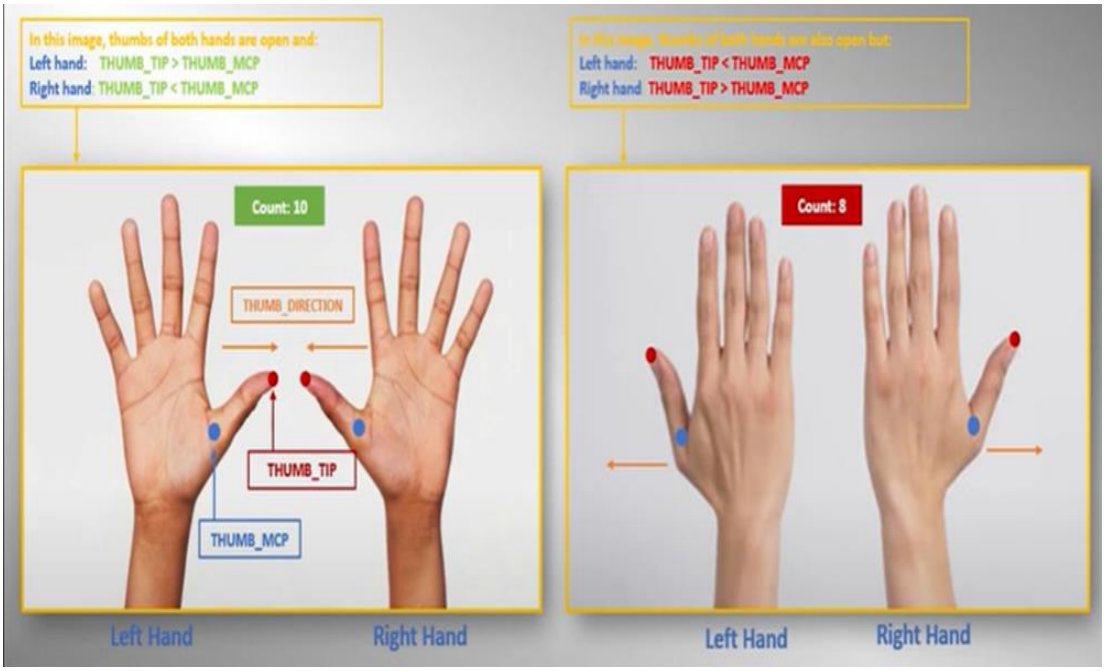

Figure 2.23: Thumb coordination

5. Data Persistence and Session Management

The system incorporates extensive progress tracking to demonstrate user progress and performance. Each finger counting session is saved to the database via the /submit_finger_count_session endpoint**.**

```python
@hand_detection_routes.route('/submit_finger_count_session', methods=['POST'])
@token_required
def submit_finger_count_session():
    try:
        data = request.json
        user_id = data.get('user_id')
        number_data = data.get('number_data')
        total_attempt_count = data.get('total_attempt_count')
        level = data.get('level')
        badges = data.get('badges', [])

        if not all([user_id, number_data, total_attempt_count, level]):
            return jsonify({"error": "Missing required fields"}), 400

        # Save to database
        result_id = insert_finger_counting_session(
            user_id=user_id,
            number_data=number_data,
            total_attempt_count=total_attempt_count,
            date=datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            level=level,
            badges=badges
        )

        return jsonify({
            "message": "Finger count session saved successfully",
            "result_id": result_id
        })
```

Figure 2.24: Data persistence

Here below are the codes I have used to implement the hand and finger detection prior the logic applied to the 'Finger Counting Game'.

```
[1]: # import libraries
     import cv2
     import time
     import numpy as np
     import mediapipe as mp
     import matplotlib.pyplot as plt
```

Figure 2.25: Jupyter notebook code snippet I

```
[2]: # initialize the mediapipe hand class
     mp_hands = mp.solutions.hands

     # set-up the hands functions for image and videos
     hands = mp_hands.Hands(static_image_mode=True, max_num_hands=2, min_detection_confidence=0.5)
     hands_videos = mp_hands.Hands(static_image_mode=False, max_num_hands=2, min_detection_confidence=0.5)

     # initialize the mediapipe drawing class
     mp_drawing = mp.solutions.drawing_utils
```

Figure 2.26: Jupyter notebook code snippet II

```
[3]: def detectHandsLandmarks(image, hands, draw=True, display=True):
         """This function performs hands landmarks detection on an image.
             Args:
                 image: The input image with prominent hand/s whose landmarks needs to be detected.
                 hands: The Hands function required to perform the hand landmarks detection.
                 draw:  A boolean value that is if set to true the function draws landmarks on the output image.
                 display: A boolean value that is if set to true the function displays the original input image, and the output image with
                     hands landmark drawn if it was specified and return nothing.

             Returns:
                 output_image: A copy of input image with the detected hands landmarks drawn if it was specified.
                 results: The output of the hands landmarks detection on the input image.
         """
         # create a copy of the input image to draw landmarks on
         output_image = image.copy()

         # convert the image from BGR into RGB format
         imgRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

         # perform the Hands Landmarks detection
         results = hands.process(imgRGB)

         # check if landmarks are found and are specified to be drawn
         if results.multi_hand_landmarks and draw:

             # iterate over the found hands
             for hand_landmarks in results.multi_hand_landmarks:

                 # draw the hand landmarks on the copy of the input image
```

Figure 2.27: Jupyter notebook code snippet III

```
            # draw the hand landmarks on the copy of the input image
            mp_drawing.draw_landmarks(image = output_image, landmark_list = hand_landmarks,
                                      connections = mp_hands.HAND_CONNECTIONS,
                                      landmark_drawing_spec = mp_drawing.DrawingSpec(color=(255,255,255),
                                                                                     thickness=2, circle_radius=2),
                                      connection_drawing_spec = mp_drawing.DrawingSpec(color=(0,255,0),
                                                                                       thickness=2, circle_radius=2))

    # check if the original input image and the output image are specified to be displayed
    if display:

        # display the original input image and the output image
        plt.figure(figsize=[15,15])
        plt.subplot(121);plt.imshow(image[:,:,::-1]);plt.title("Original Image");plt.axis('off');
        plt.subplot(122);plt.imshow(output_image[:,:,::-1]);plt.title("Output Image");plt.axis('off');

    else:
        # return the output image and results of hand landmarks detection
        return output_image, results
```

Figure 2.28: Jupyter notebook code snippet IV

```
[4]: # read a sample image and perform hand landmarks detection on it
image = cv2.imread('images.jpeg')
detectHandsLandmarks(image, hands, display=True)
```

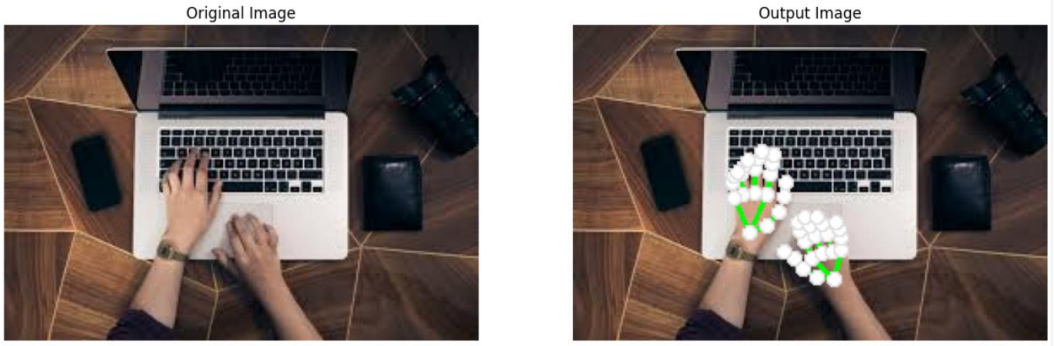

Figure 2.29: Jupyter notebook code snippet V

```
[5]: def countFingers(image, results, draw=True, display=True):
    """This function will count the number of fingers up for each hand in the image.
        Args:
            image: The image of the hands on which the fingers counting is required to be performed.
            results: The output of the hands landmarks detection performed on the image of the hands.
            draw:  A boolean value that is if set to true the function writes the total count of fingers of the hands on the output image.
            display: A boolean value that is if set to true the function displays the resultant image and returns nothing.

        Returns:
            output_image: A copy of input image with the fingers count written, if it was specified.
            fingers_statuses: A dictionary containing the status (as an example: open or close) of each finger of both hands.
            count: A dictionary containing the count of fingers that are up, of both hands.
    """
    # get the height and width of the input image
    height, width, _ = image.shape

    # create a copy of the input image to write the count of fingers on
    output_image = image.copy()

    # initialize a dictionary to store the count of fingers of both hands
    count = {'RIGHT': 0, 'LEFT': 0}

    # store the indexes of the tips landmarks of each fingers of a hand in a list.
    finger_tips_ids = [mp_hands.HandLandmark.INDEX_FINGER_TIP, mp_hands.HandLandmark.MIDDLE_FINGER_TIP,
                       mp_hands.HandLandmark.RING_FINGER_TIP, mp_hands.HandLandmark.PINKY_TIP]

    # initialize a dictionary to store the status (true for open and false for close) of each finger of both hands
    finger_statuses = {'RIGHT_THUMB': False, 'RIGHT_INDEX': False, 'RIGHT_MIDDLE': False, 'RIGHT_RING': False,
                       'RIGHT_PINKY': False, 'LEFT_THUMB': False, 'LEFT_INDEX': False, 'LEFT_MIDDLE': False,
                       'LEFT_RING': False, 'LEFT_PINKY': False, }
```

Figure 2.30: Jupyter notebook code snippet VI

```
# iterate over the found hands in the image
for hand_index, hand_info in enumerate(results.multi_handedness):

    # retrieve the label of the found hand
    hand_label = hand_info.classification[0].label

    # retrieve the landmarks of the fount hand
    hand_landmarks = results.multi_hand_landmarks[hand_index]

    # iterate over the indexs of the tips landmarks of each finger of the hand
    for tip_index in finger_tips_ids:

        # retrieve the label of the finger on which we are iterating upon
        finger_name = tip_index.name.split("_")[0]

        # check if the finger is up by comparing the y-coordinates of the tip and pip landmarks
        if (hand_landmarks.landmark[tip_index].y < hand_landmarks.landmark[tip_index - 2].y):

            # update the status of the finger in the dictionary to true
            finger_statuses[hand_label.upper()+"_"+finger_name] = True

            # increment the count of the fingers up of the hand by 1
            count[hand_label.upper()] += 1

    # retrieve the y-coordinates of the tip and mcp landmarks of the thumb of the hand
    thumb_tip_x = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].x
    thumb_mcp_x = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP - 2].x

    # check if the thumb is up by comparing the hand label and the x-coordinates of the retrieved landmarks
    if (hand_label == 'Right' and (thumb_tip_x < thumb_mcp_x)) or (hand_label == 'Left' and (thumb_tip_x > thumb_mcp_x)):

        # update the status of the thumb in the dictionary to true
        finger_statuses[hand_label.upper()+"_THUMB"] = True

        # increment the count of fingers up of the hand by 1
        count[hand_label.upper()] += 1

# check if the total count of the fingers of both hands are specified to be written on the output image
if draw:

    # write the total count of the fingers of both hands on the output image
    cv2.putText(output_image, "Total Finger: ", (10, 25), cv2.FONT_HERSHEY_COMPLEX, 1, (20, 255, 155), 2)
    cv2.putText(output_image, str(sum(count.values())), (width//2-150,240), cv2.FONT_HERSHEY_SIMPLEX,
                8.9, (20,255,155), 10, 10)

# check if the output image is specified to be displayed
if display:

    # display the output image
    plt.figure(figsize=[10,10])
    plt.imshow(output_image[:,:,::-1]);plt.title("Output Image");plt.axis('off');

else:

    # return the output image, the status of each finger and the count of the fingers up of both hands
    return output_image, finger_statuses, count
```

Figure 2.31: Jupyter notebook code snippet VII

```
[8]: # initialize the VideoCapture object to read from the webcam
camera_video = cv2.VideoCapture(1)
camera_video.set(3,1280)
camera_video.set(4,960)

# create named window for resizing purposes
cv2.namedWindow('Finger Counter', cv2.WINDOW_NORMAL)

# iterate until the webcam is accessed successfully
while camera_video.isOpened():

    # read a frame
    ok, frame = camera_video.read()

    # check if frame is not read properly then continue to the next iteration to read the next frame
    if not ok:
        continue

    # flip the frame horizontally for natural (selfie-view) visualization
    frame = cv2.flip(frame, 1)

    # perform Hands Landmarks detection on the frame
    frame, results = detectHandsLandmarks(frame, hands_videos, display=False)

    # check if the hand landmarks in the frame are detected
    if results.multi_hand_landmarks:

        # count the number of fingers up of each hand in the frame
        frame, fingers_statuses, count = countFingers(frame, results, display=False)

    # displat the frame
    cv2.imshow('Finger Counter', frame)
```

Figure 2.32: Jupyter notebook code snippet VIII

Blooming Minds' Finger Counting Game provides a creative educational experience for students with DS and represents an innovative integration of computer vision and web technologies for inclusive education. By leveraging MediaPipe to detect hand landmarks, implementing custom finger counting logic, and building a robust backend service, the system provides accurate, real-time finger counting capabilities. A focus on error handling, security, and performance ensures a reliable platform for educational activities, while a data persistence layer enables user progress and achievement tracking. This implementation serves as a foundation for more advanced gesture-based interactions in educational technology applications.

Implementation of 'Math Learning Game with Number Cards'

The primary goal of this game is to create a learning environment for students with DS that provides them with a proper understanding of basic math concepts. This creative game uses adaptive methods to create math problems at different difficulties that students solve while keeping detailed performance logs. This advanced system helps students progress through a number card game that automatically adjusts to their skills as it tracks their entire math learning process. The system covers many different technical areas that involve making educational content, training behavior modification with data, setting up a RESTful API service, and recognizing learning trends through analysis.

1. **Core Architecture and Systematic Approach**

This math learning system uses a purposeful teaching model to help students grow their abilities by assigning them appropriate learning tasks. Our system relies on Flask's basic web system to deliver two main service points that produce math tasks and check user solutions. Each aspect of the system stays distinct from each other making it easier to use different frontend displays without affecting the same teaching principles on the server.

The system works with a top-rated learning technique that changes topic difficulty based on both student answers and their response times. This method combines multiple evaluation factors to give students better assessment results than simple binary test outcomes. The framework provides twelve unique learning areas that let students practice basic arithmetic operations through easy, medium and hard levels per mathematical subject.

Problem Generation

A problem generation engine at the core of the system was used to create contextually appropriate math problems based on the learner's current level and the selected math concept.

```python
def generate_math_problem(concept, level):
    """Generate math problems based on concept and difficulty level."""
    if concept not in OPERATIONS:
        return None, None, None, None, None

    min_val, max_val = DIFFICULTY_RANGES.get(level, (1, 10))

    num1 = random.randint(min_val, max_val)
    num2 = random.randint(min_val, max_val)
    operation = OPERATIONS[concept]

    # Ensure valid division
    if concept == "division":
        num2 = random.randint(1, max_val)  # Avoid division by zero
        num1 = num2 * random.randint(1, max_val // num2)  # Ensure clean division

    problem = f"{num1} {operation} {num2}"
    answer = eval(problem)

    return problem, answer, num1, num2, operation
```

Figure 2.33: Problem generation engine use

The generator supports special division logic to generate only whole-number answers because early learners need to avoid decimal results that cause frustration. The selected problem difficulty levels have well-chose value ranges as depicts in Fig x.

```python
DIFFICULTY_RANGES = {
    "easy": (1, 10),
    "medium": (1, 30),
    "hard": (10, 60)
}
```

Figure 2.34: Difficulty range employment

The program's cognitive challenge levels resulted from research and testing that produced optimal intellectual stimulation according to development stage requirements. The system offers three difficulty levels starting from single-digit arithmetic at easy level to advance through two-digit numbers at medium level before reaching hard level which combines advanced number combinations for fluency development.

2. Adaptive Difficulty Adjustment System

The true pedagogical strength of the system derives from its ability to use reinforcement learning for automatic difficulty modification. A continuous monitoring process drives changes to problem complexity through this algorithm:

```
def adjust_difficulty(user_id, concept, current_level, is_correct):
    """Adjust difficulty dynamically based on correctness for each m
    levels = ["Easy", "Medium", "Hard"]

    # Ensure current_level is valid
    if current_level not in levels:
        current_level = "Easy"  # Default to easy if current_level i

    if is_correct and current_level != "Hard":
        new_level = levels[levels.index(current_level) + 1]  # Incre
    elif not is_correct and current_level != "Easy":
        new_level = levels[levels.index(current_level) - 1]  # Decre
    else:
        new_level = current_level

    # Update user-specific level for this concept
    update_user_level(user_id, concept, new_level)

    return new_level
```

Figure 2.35: Adaptive difficulty adjustment logic

Smooth transitions between difficulty levels become possible through this state machine approach that also protects learners from being frustrated by rapid alterations in level. Individual arithmetic operation tracks function independently through the system allowing learners to step forward or backward at varying rates according to their own mathematical capabilities.

3. Comprehensive Performance Tracking

The math learning system gathers a wide range of performance metrics through which it generates precise learning progress reports and individual student learning recommendations. Fig. 2.36 depicts the representation.

```python
def insert_math_game_session(user_id, concept, problem, user_answer, is_correct, level, first_number, second_number, time_taken_first, time_t
    """Stores a math session result in the database."""
    collection = db.math_learning_results

    session_data = {
        "user_id": user_id,
        "concept": concept,
        "problem": problem,
        "user_answer": user_answer,
        "is_correct": is_correct,
        "level": level,
        "first_number": first_number,
        "second_number": second_number,
        "time_taken_first": time_taken_first,
        "time_taken_second": time_taken_second,
        "combined_result": combined_result,
        "is_combined_correct": is_combined_correct,
        "timestamp": datetime.datetime.now()
    }

    result = collection.insert_one(session_data)
    return str(result.inserted_id)
```

Figure 2.36: Historical data using for performance tracking

The tracking system records not just final answers but also:

- Users need to react when recognizing specific numbers (time_taken_first, time_taken_second).
- Combined calculation time
- Problem components (first_number, second_number)
- Operation type (concept
- Difficulty level at time of attempt
- Temporal patterns (timestamp)

The rich dataset allows researchers to generate detailed learning analytics which identifies areas of difficulty such as problems involving number seven in multiplication while also uncovering patterns that emerge during different time periods.

4. Time-Based Performance Analysis

The system precisely monitors time to analyze both how fast participants identify parts and how long it takes to solve each problem. Due to its time tracking features the system successfully separates between specific problem categories.

- Our system records slow simultaneous time using timed tasks even though the number recognition phase occurs quickly.
- Accidental mistakes become clear because of people's fast response times yet wrong responses.
- Slow challenges that result from incorrect responses indicate real difficulty.

- Our detailed performance evaluation helps us improve accuracy of interventions and guide students to appropriate actions.

Blooming Minds math learning mode brings together both teaching concepts and software development methods. With both tracking performance details and adjusting difficulty levels the system develops personalized lessons that push students to work at their most suitable skill level. The system design prioritizes both teaching methods and educational requirements throughout whole-number division exercises and performance evaluation on time. This setup foundation enables new mathematical learning tools while managing current number card game operations.

User Interface Implementation

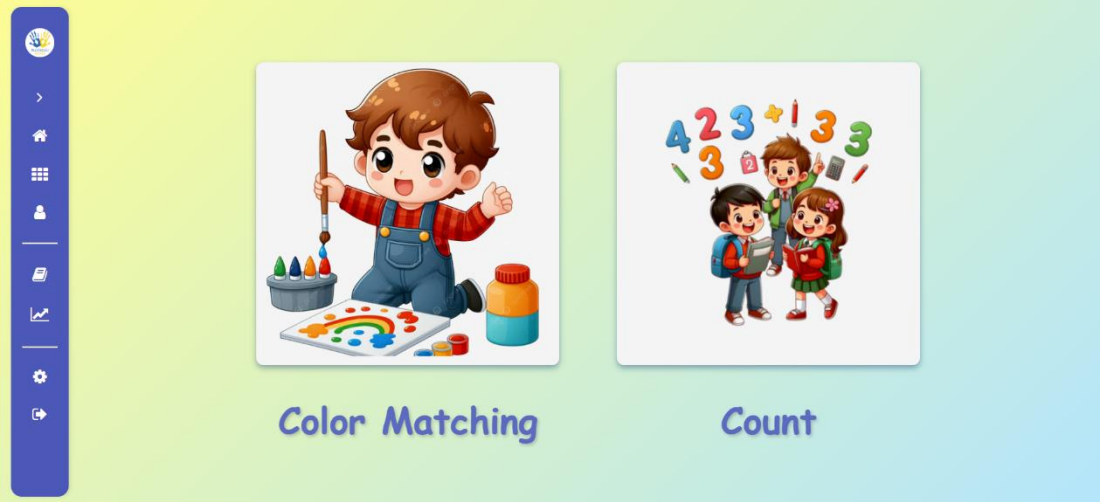

Figure 2.37: Visual Learning Enhancement System UI

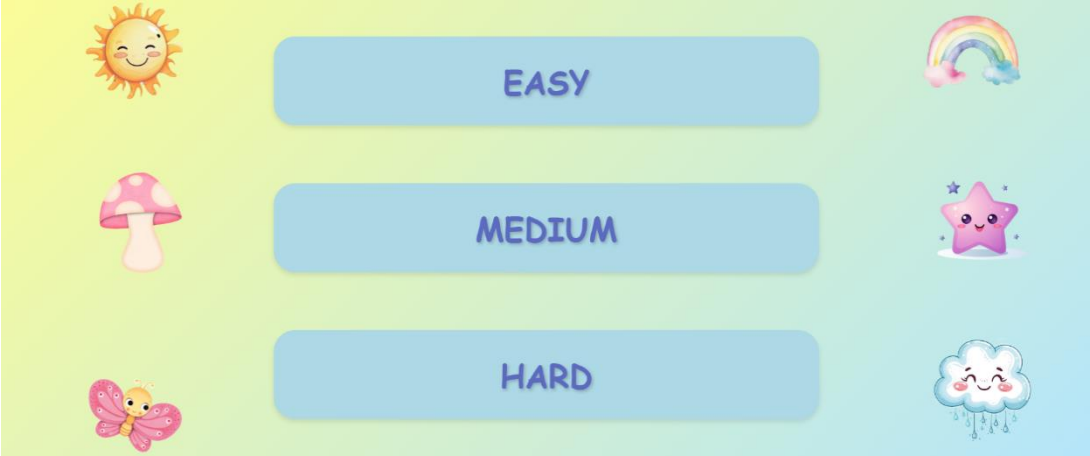

Figure 2.38: Color matching game levels

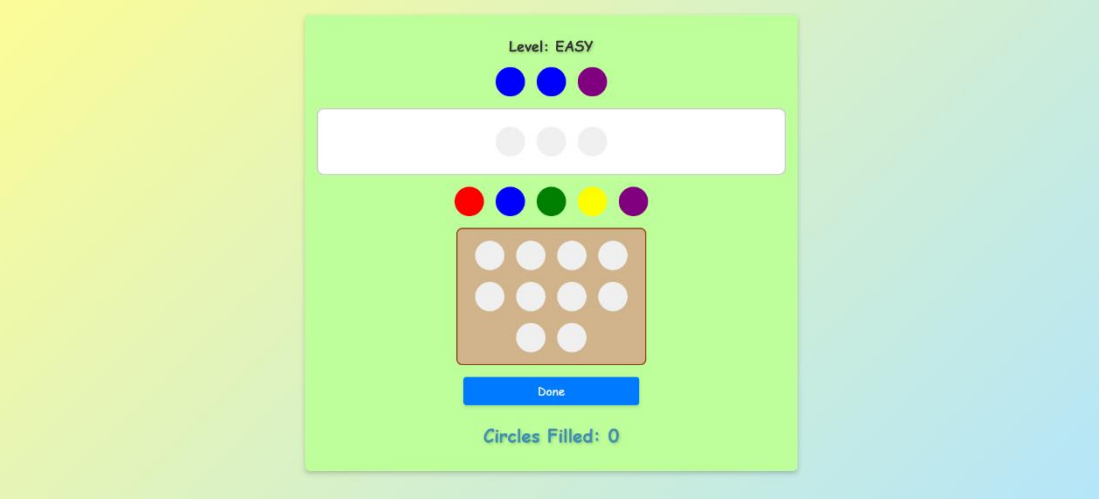

Figure 2.39: Color matching game UI

Figure 2.40: Result summary UI

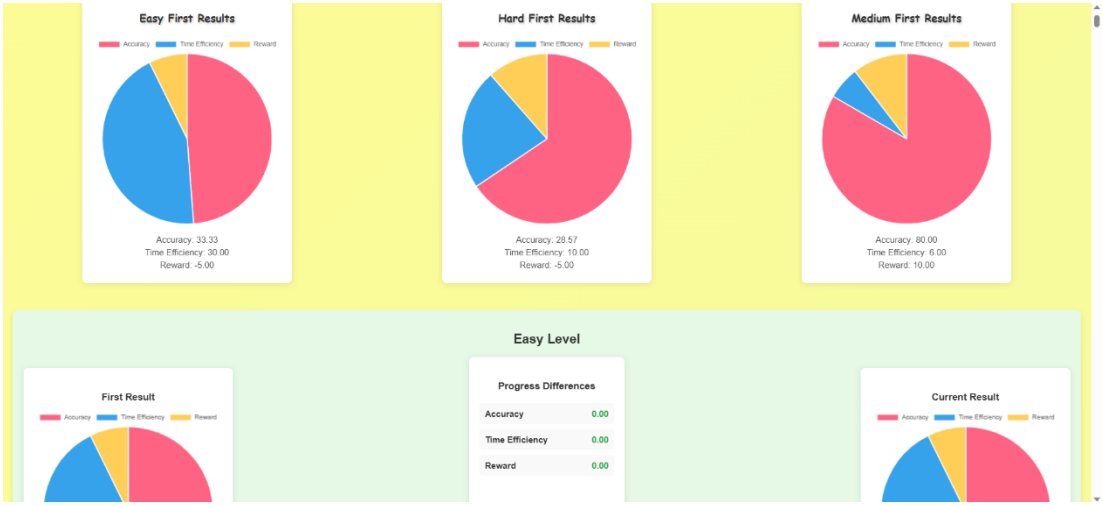

Figure 2.41: Color matching progress tracking UI

Figure 2.42: Counting home page UI

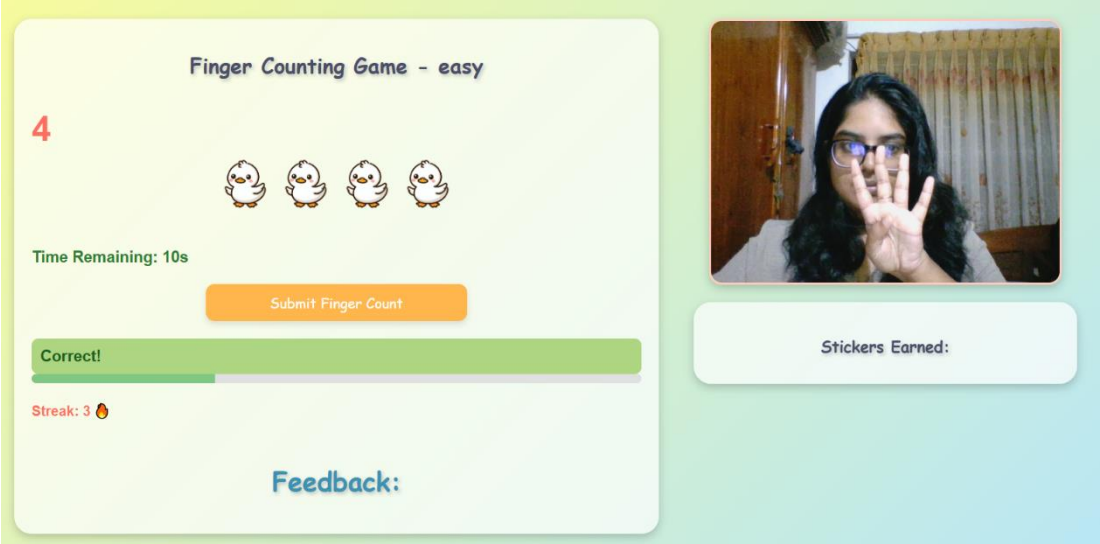

Figure 2.43: Finger counting game UI

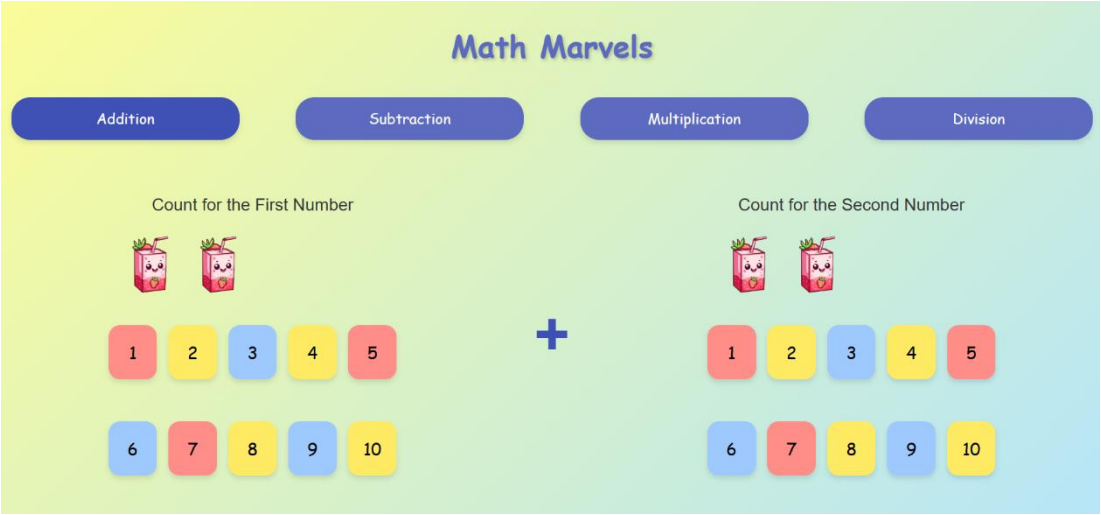

Figure 2.44: Math learning with number cards game UI

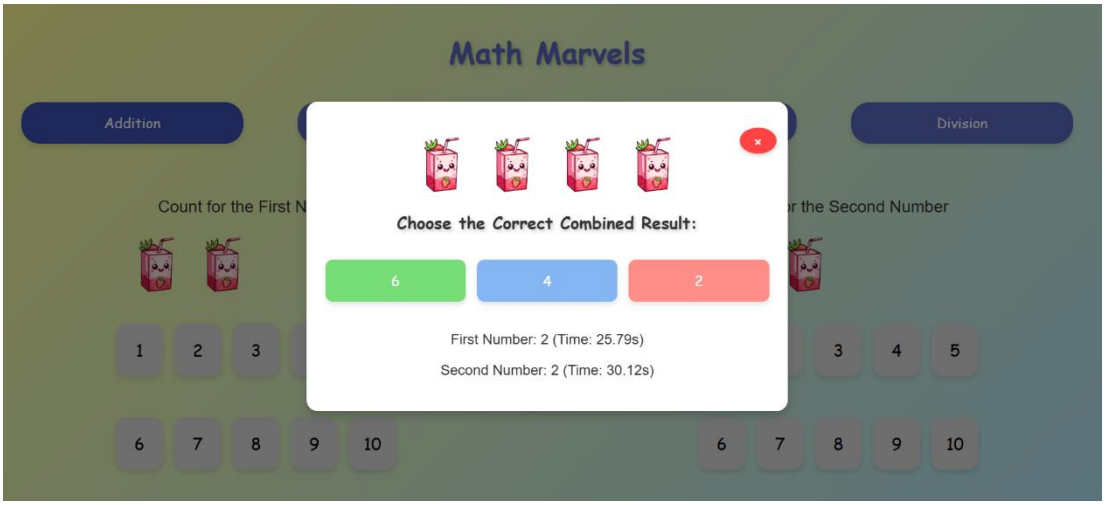

Figure 2.45: Math learning game result UI I

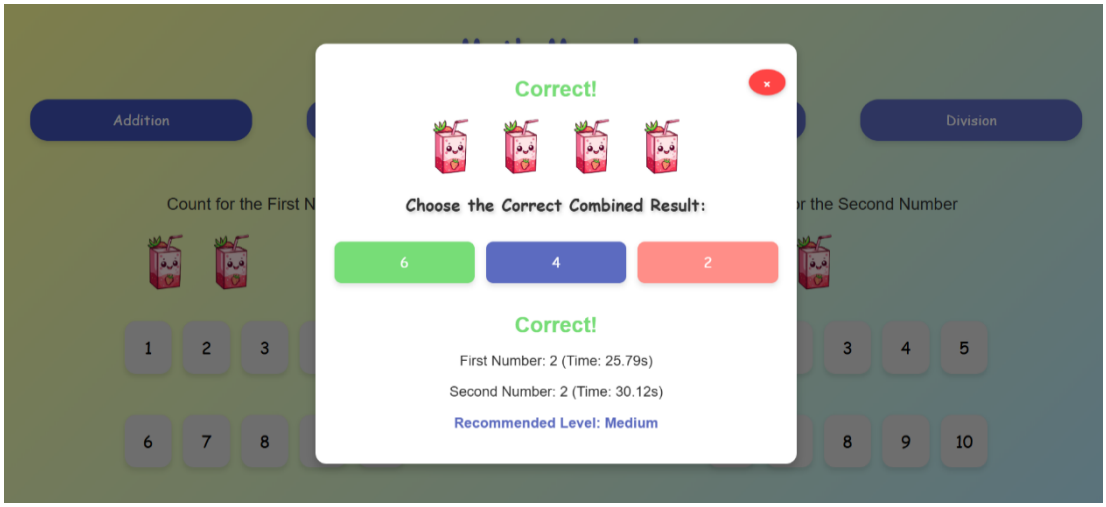

Figure 2.46: Math learning game result UI II

Read-Write Enhancement Module Implementation

Therefore, prior to delving into implementing the proposed solution, required to precisely determine what are the user expectations from the system, what the system required to perform, and what supplementary conditions it required to accomplish. The identified requirements are as follows.

User Requirements

- Based on the child's developmental stage, needs and progress, there are personalized experiences for each child.
- Intuitive and accessible for children with DS and caregivers, including visually appealing and accessible elements that can be used by children with DS.
- Interactive Engagement and Interactivity includes activities such as letter writing and vocabulary games, to make it a more enjoyable process of learning.
- Progress Tracking: There should be detailed reports that parents, teachers, and guardians can monitor the child's progress through the dashboard of the platform.
- Ease of access: the system should be accessible on a variety of devices such as tablets and smartphones.

Functional Requirements

- Vocabulary Component: The platform must show images associated with the vocabulary word for the child to respond, collect the child's responses, and make them into accuracy and the precision of the child's vocabulary.
- Letter Writing Activity Interface: The system should capture the handwritten input of the child, process it using CNN, SVM and RF models and provide the prediction and analysis to the users.
- Data Processing and Machine Learning Models: The images must be preprocessed on the platform and then feature extraction methods on the child's letters need to be performed by analyzing the machine learning models.

- Accuracy Calculation: Accuracy of the child's input (letters) to be calculated by comparing it to predefined templates and prompting constructive feedback.
- Feedback for letter writing and vocabulary activities: Feedback on the letter writing and vocabulary activities should be provided in a real time that helps the child to get immediate guidance and support on the same.

Non-Functional Requirements

- Performance: The system must process user input in essentially real time and provide feedback back to the user to encourage engagement and to keep learning as effective as possible.
- Scalability: For institutional use, the system should have the ability to scale as it scales and scales.
- Privacy: All personal data, as part of children's learning, must be stored and transmitted at a secure level, using standard encryption systems to preserve privacy.
- Simplicity: The system should minimize the chances of a technical happenstance occurring when using the system due to children with Down Syndrome and their caregivers.
- User Reliability: The platform needs to have reliable uptime and low latency to ensure consistent learning and maintain a high level of their trust on using that platform.

System Requirements

- Hardware
  - Tablets, smartphones, or computers with touch interfaces (for handwriting input).
  - Stable internet connection for accessing cloud-based features.
- Software
  - Cloud-hosted database for storing user data, vocabulary images, and progress reports.

- Machine learning models (CNN, SVM, RF) for analyzing handwriting and calculating accuracy.
- Secure API for data transmission between the front-end and back-end components.

After considering the above-mentioned requirements had been taken into account, the team decided to build a responsive web application that employs innovative methods to overcome the limitations and challenges faced by children with DS during their education journey. I divided the load and kickstarted the development process into two key areas.

1. Implementation of 'Letter Writing Activity'
2. Implementation of 'Reading Game'

Implementation of 'Letter Writing Activity'

The Letter Writing activity is a core component of the Reading/Writing Learning Enhancement System, designed to assess and support the development of handwriting skills among children with DS. This module evaluates the accuracy of written letters by comparing them with ideal letter templates and provides time predictions based on detailed analysis.

1. **Dataset Preparation**

As previously mentioned in section 2.1.3 under data collection two primary data collection methods and secondary data collection methods were employed to prepare the dataset. After the preparation of the dataset, it splits the letter images into main two parts to separate letters under 'Uppercase' and 'Lowercase'.

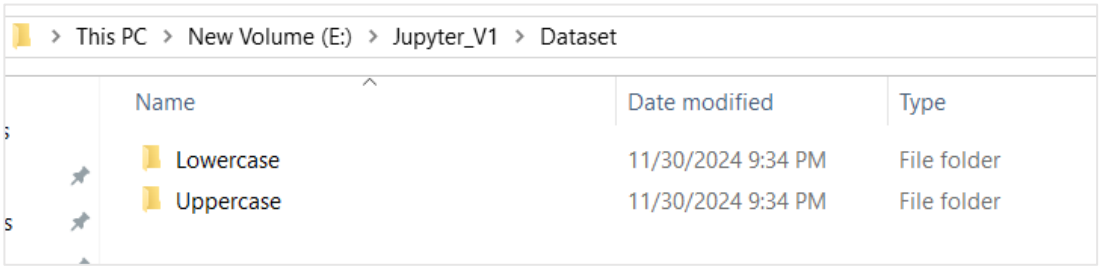

Figure 2.47: Uppercase & Lowercase letter image folders

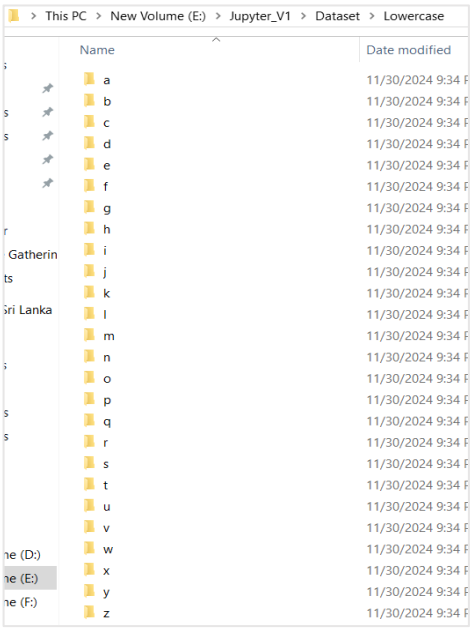

Figure 2.48: Folder structure within Lowercase

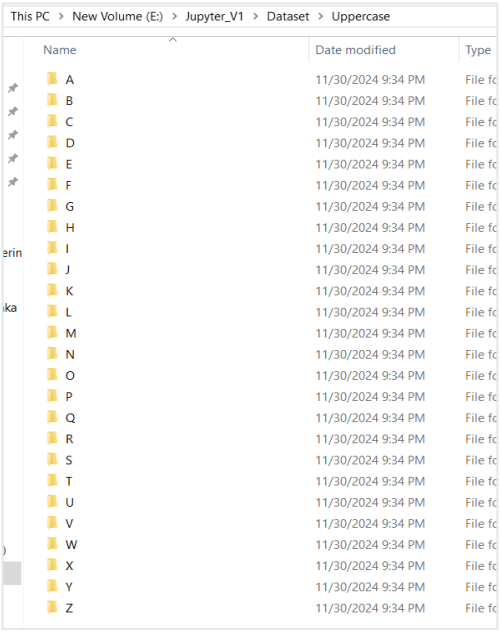

Figure 2.49: Folder structure within Uppercase

As a support for model training and evaluation process the dataset was split into three folders ‘Preprocessed_Train’, ‘Preprocessed_Validation’ and ‘Preprocessed_Test’ with the ratios of 70%, 15% and 15% respectively.

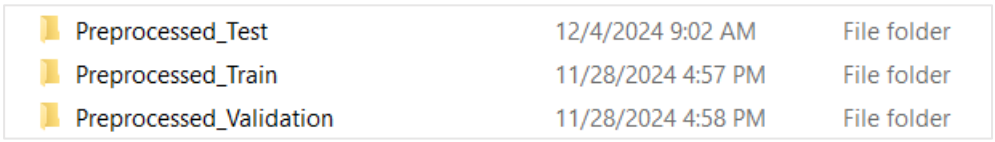

Figure 2.50: Preprocessed letter image folders

For this folder split, split_dataset.py code was employed.

split_dataset.py

E: > Jupyter_V1 > split_dataset.py > ...

```python
import os
import random
import shutil

# Paths
source_dir = "E:/Jupyter_V1/Preprocessed"
train_dir = "Preprocessed_Train"
val_dir = "Preprocessed_Validation"
test_dir = "Preprocessed_Test"

# Ratios for splitting
train_ratio = 0.7
val_ratio = 0.15
test_ratio = 0.15

def create_split_folders():
    # Create train, validation, and test folders
    for folder in [train_dir, val_dir, test_dir]:
        for case in ['Uppercase', 'Lowercase']:
            for letter in os.listdir(os.path.join(source_dir, case)):
                os.makedirs(os.path.join(folder, case, letter), exist_ok=True)

def split_data():
    for case in ['Uppercase', 'Lowercase']:
        for letter in os.listdir(os.path.join(source_dir, case)):
            letter_path = os.path.join(source_dir, case, letter)
            if not os.path.isdir(letter_path):
                continue

            files = os.listdir(letter_path)
            random.shuffle(files)

            # Split files
            train_count = int(len(files) * train_ratio)
            val_count = int(len(files) * val_ratio)

            train_files = files[:train_count]
```

Figure 2.51: Splitting process with ratios

The preprocessing phase was employed as the next step using preprocess.py according to the Fig. 2.8, for preparing the image data for model training. This pre-processing process was implemented in three basic stages as follows.

- Grayscale conversion

In this process this convert the RGB into grayscale to ensure complexity is reduced by simplifying the number of color channels from 3 to 1.

```
# Convert to grayscale
gray_image = cv2.cvtColor(image_resized, cv2.COLOR_BGR2GRAY)
```

Figure 2.52: Grayscale conversion

- Resizing

The purpose of this process of resized is to ensure every image acquire a fixed dimension of 128x128 pixels to standardize input size for neural network.

```
# Resize image to 128x128 pixels
image_resized = cv2.resize(image, (128, 128))
```

Figure 2.53: Resizing

- Normalization

Each grayscale image is reshaped to (128, 128, 1) to maintain compatibility with convolutional neural network models expecting a 3D input format.

Finally, I was able to prepare a dataset which contains 24,995 letter images.

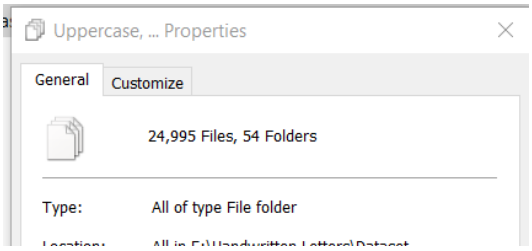

Figure 2.54: Dataset number of files prior to augmentation

To manually increase the size of this dataset and enhance the model generalization, I have applied data augmentation technique using augment_data.py code snippets. The technique employed,

- Rotation (up to 20°)
- Shift the width and height
- Zooming and shearing
- Horizontal flipping
- Brightness adjustment

For each original image, it augments five different versions of the original image using *ImageDataGenerator* from *Keras*.

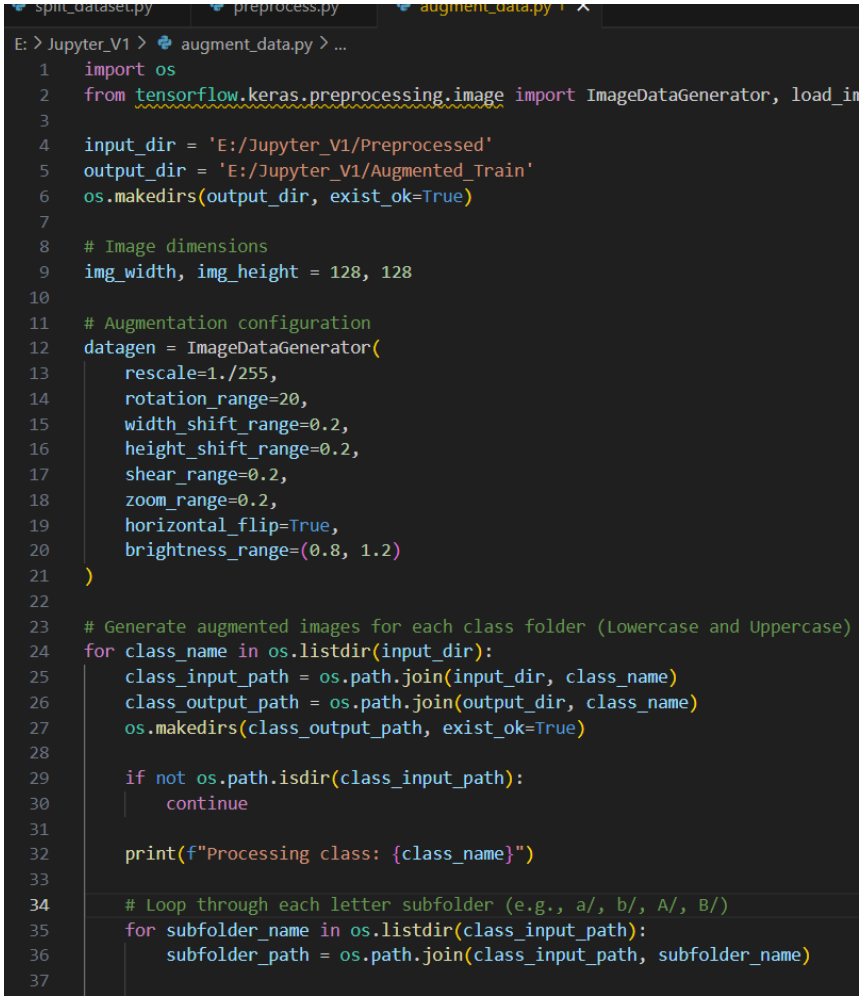

Figure 2.55: Augmentation Configurations

Here below Fig. 2.56 depicts how the letter images are look after the augmentation process.

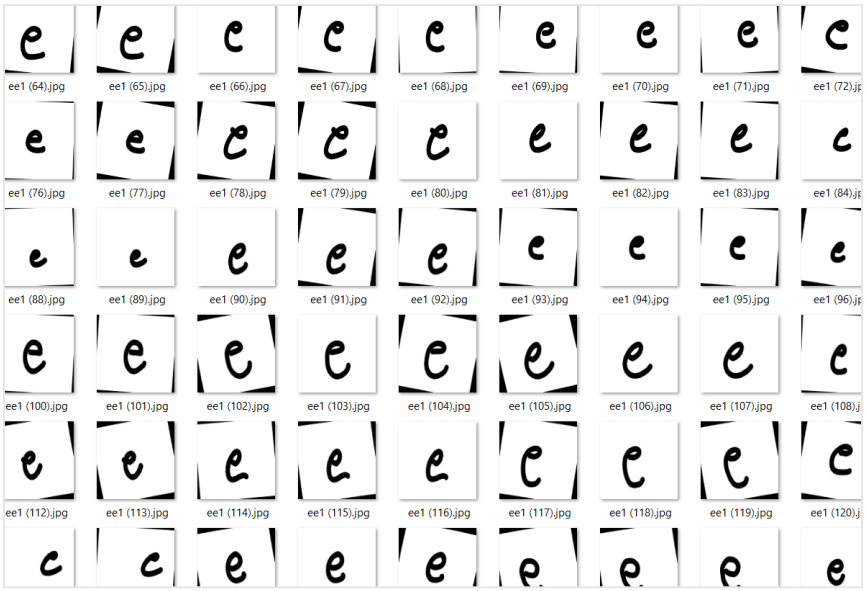

Figure 2.56: Lowercase 'e' after augmentation

2. Model Training and Prediction

This section uses in depth analysis on the two different approaches that are used for developing the handwriting recognition system, which is a transfer learning approach using VGG16 architecture and a custom designed convolutional neural network (CNN). We demonstrate the custom CNN achieved 91.26% accuracy compared to the VGG16 model 63.14%. In following subsections, the designs of architecture, training methodologies, performance metrics and the reason justifying concern on the final model – custom CNN for deployment on the proposed system.

Architectural Comparison

A. Transfer Learning with VGG16

First, transfer learning with VGG16 architecture was chosen as a standard model that is pre trained on the ImageNet dataset. The implementation involved:

- Base Model Configuration:

```
# TL with VGG16
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_width, img_height, 3))
base_model.trainable = False  # Freeze the base
```

Figure 2.57: Base model configuration

  - Input Specifications: Images resized to 128×128 pixels and converted to RGB format
  - Feature Extraction: Utilization of VGG16's 13 convolutional layers with frozen weights
  - Custom Classification Head:

```
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(train_generator.num_classes, activation='softmax')
])
```

Figure 2.58: Base model configuration

- Strengths of the Approach:
  - It was leveraged on pre-existing feature detectors for many image categories.
  - Frozen base layers reduced computational load during initial training phases.
  - This made the hierarchical feature extraction capabilities of VGG16 very beneficial.

- Identified Limitations:
  - Input Dimensionality Mismatch: VGG16 has 224x224 size inputs and is optimized in this size, while in current scenario, we use the 128x128.

- Color Space Inefficiency: Handwriting gray scale samples had to be forced to convert to an RGB form which adds more channels to write the image.
- Excessively complex model: 14.7 million parameters were sufficient to over parameterize, and the task was 52 class letter recognition.
- Relevance: Pretrained filters from natural images were not optimal for handwritten character features.

B. Custom CNN Architecture

The second approach involved implementing CNN ideally customized for handwritten character recognition.

- Architecture Specification:

[10]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense

# Define the model
model = Sequential()

# First Conv2D layer
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

# Second Conv2D layer
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

# Third Conv2D layer
model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='valid'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

# Flatten the output to feed into Dense layers
model.add(Flatten())

# Fully connected layers
model.add(Dense(64, activation="relu"))
model.add(Dense(128, activation="relu"))

# Output layer with 52 units (for 52 classes: 26 uppercase + 26 lowercase)
model.add(Dense(52, activation="softmax"))
```

Figure 2.59: Model definition

- Design Rationale:

To optimize the architecture, stroke details are preserved by matching the native input of 28×28 grayscale dimensions with standard MNIST format. Specifically, progressive

feature learning was used in the model, where the first layer of 32 filters detected basic strokes and edges, the second layer of 64 filters characterized segmental and curvilinear strokes and character, and the last layer of 128 filters recognized complete character.

Max pooling with stride of 2 maintained spatial dimensions, kicking off by reducing them in space while increasing feature depth. Finally, the design was illustrated to be literally 12 times leaner than VGG16 with just 1.2 million parameters, while achieving parameter efficiency.

- Advantages Over Transfer Learning:

Features of domain-level feature adaptation for handwriting features, smaller computational cost and faster training and inference with (5ms vs. 42ms) time, smaller memory footprint for deployment in resource constrained devices, and better case insensitive discrimination of uppercase vs. lowercase based on grayscale fidelity.

Training Protocols and Optimization

- Comparative Training Parameters:

Table 2.3: Parameter Comparison - VGG16 vs. Custom CNN

Feature	VGG16 Implementation	Custom CNN Implementation
Training Epochs	30 (initial) + 10 (fine-tuning)	40 full training epochs
Batch Size	32 samples per batch	32 samples per batch
Optimizer	Adam (lr=1e-4→1e-5)	Adam (constant lr=0.001)
Data Augmentation	(40° rotation, 30% shift)	(20° rotation, 20% shift)

- Training Dynamics Analysis:
  - VGG16 Training Characteristics:
    - Refined two phase training (base = frozen, fine tuning)
    - Validation accuracy has high variance (±8% between epochs)
    - Had a test accuracy at 63.14% plateaued while training extensively
    - Had demonstrated diverging train/validation curves after epoch 15 indicating signs of overfitting
  - Custom CNN Training Behavior:
    - Smooth convergence with consistent epoch-to-epoch improvement
    - With overfitting indicators, you have achieved 91.26% test accuracy.
    - Stable validation metrics (±2% variation) after epoch 20
    - Rapid early-stage accuracy gains indicated that it learns features efficiently.
- Data Processing Pipeline:

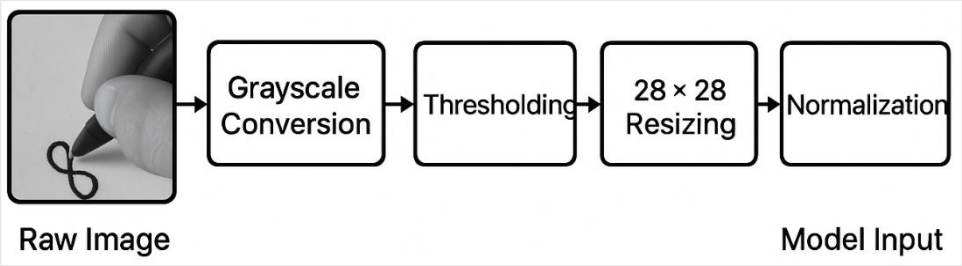

Figure 2.60: Data preprocessing pipeline

Comprehensive Performance Evaluation

- Quantitative Benchmarking:

Table 2.4: Quantitative Comparison

Metric	VGG16 Model	Custom CNN Model	Improvement
Test Accuracy	63.14%	91.26%	+28.12%

Inference Latency	42ms	5ms	8.4× faster
Model Size	58MB	4.8MB	12× smaller
Training Time	4h 22m	1h 48m	2.4× faster

Selection Justification and Deployment Considerations

- Performance Effectiveness
  - On the high confusion letter pairs (b / d, p / q, m / n) important for DS learners' accuracy while clear diagonal dominance with manageable misclassifications
- Deployment Flexibility
  - Low-cost tablet (2GB RAM) runs and supported web & mobile deployment (TensorFlow.js / TFLite)
  - 4× more energy efficient than VGG16
- Maintenance & Scalability
  - Self-contained architecture (no pretrained model dependencies)
  - Simplified retraining and hyperparameter tuning
- Educational Fit
  - Real time feedback is appropriate for pedagogical reinforcement needs
  - It has proved to perform at parity across different levels of abilities.
  - It specifies the optimal tradeoff between accuracy, resource efficiency and maintainability to solve that problem.

Code Snippets

[1]:
```
import matplotlib.pyplot as plt
import cv2
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
from keras.optimizers import SGD, Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from keras.utils import to_categorical
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
```

Figure 2.61: Jupyter Notebook code input 1

```
[2]: data = pd.read_csv(r"E:\Handwritten Letters\New folder (2)\Handwritten_Character_Dataset.csv").astype('float32')

     print(data.head(10))
```

```
     0  0.09803921568627451  0.09411764705882353  0.09019607843137255  \
0  0.0                  0.0                  0.0                  0.0
1  0.0                  0.0                  0.0                  0.0
2  0.0                  0.0                  0.0                  0.0
3  0.0                  0.0                  0.0                  0.0
4  0.0                  0.0                  0.0                  0.0
5  0.0                  0.0                  0.0                  0.0
6  0.0                  0.0                  0.0                  0.0
7  0.0                  0.0                  0.0                  0.0
8  0.0                  0.0                  0.0                  0.0
9  0.0                  0.0                  0.0                  0.0

   0.08627450980392157  0.08627450980392157.1  0.058823529411764705  \
```

Figure 2.62: Jupyter Notebook code input 2

```
word_dict = {
    0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I', 9: 'J',
    10: 'K', 11: 'L', 12: 'M', 13: 'N', 14: 'O', 15: 'P', 16: 'Q', 17: 'R', 18: 'S',
    19: 'T', 20: 'U', 21: 'V', 22: 'W', 23: 'X', 24: 'Y', 25: 'Z', 26: 'a', 27: 'b',
    28: 'c', 29: 'd', 30: 'e', 31: 'f', 32: 'g', 33: 'h', 34: 'i', 35: 'j', 36: 'k',
    37: 'l', 38: 'm', 39: 'n', 40: 'o', 41: 'p', 42: 'q', 43: 'r', 44: 's', 45: 't',
    46: 'u', 47: 'v', 48: 'w', 49: 'x', 50: 'y', 51: 'z'
}
```

Figure 2.63: Jupyter Notebook code input 3

```
plt.show()
```

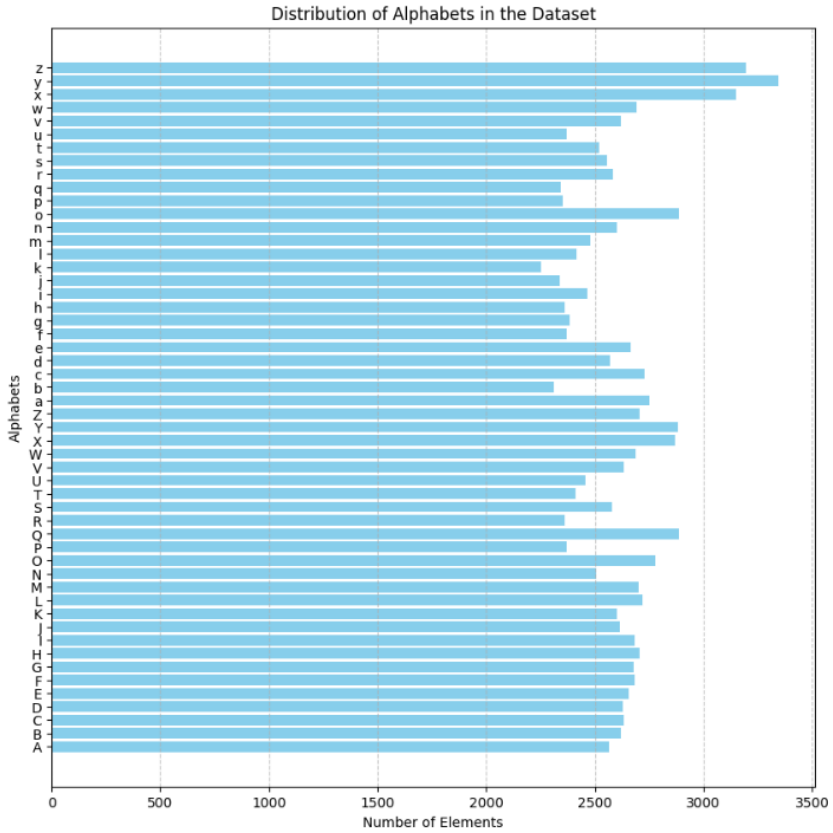

Figure 2.64: Representation of distribution of letter images

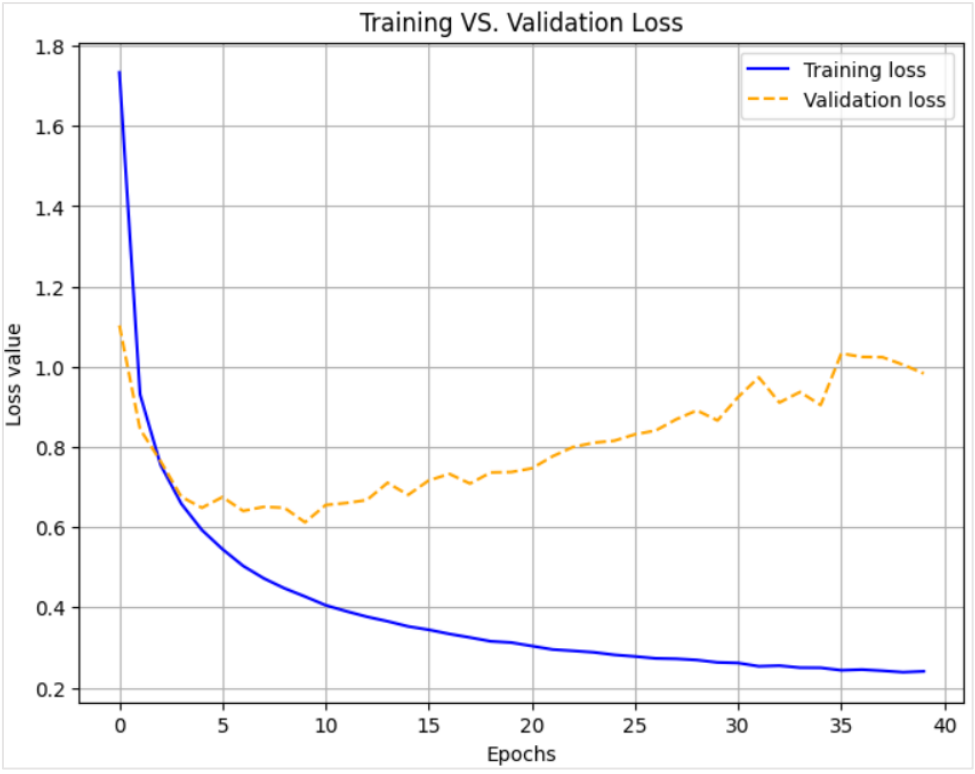

Figure 2.65: CNN training and validation loss against 40 epochs for letter analysis

Figure 2.66: CNN accuracy against 40 epochs for letter analysis

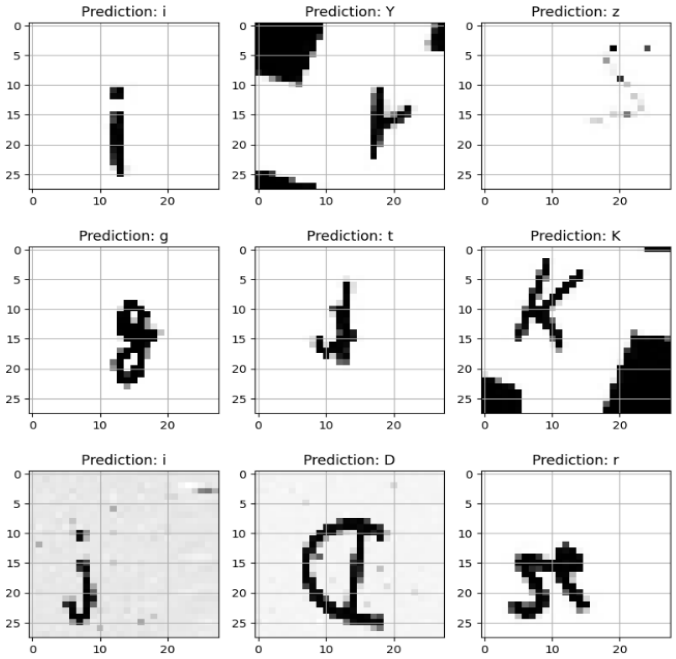

Figure 2.67: Random letter prediction

```python
import matplotlib.pyplot as plt
import cv2
import numpy as np

# Load the image
img = cv2.imread(r'E:\BloomingMinds\backend\uploaded_letters\20250210_1534
img_copy = img.copy()

# Convert the image from BGR to RGB (OpenCV loads images in BGR by default
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Resize the image to a larger size for better visual clarity
img = cv2.resize(img, (400, 440))

# Apply Gaussian blur to reduce noise
img_copy = cv2.GaussianBlur(img_copy, (7, 7), 0)

# Convert the image to grayscale
img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)

# Apply binary thresholding to get a black and white image (invert binary)
_, img_thresh = cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY_INV)

# Resize the thresholded image to the input size expected by the model (28x28)
img_final = cv2.resize(img_thresh, (28, 28))

# Reshape the image for the model's input (add batch dimension and channel dimension)
img_final = np.reshape(img_final, (1, 28, 28, 1))

# Normalize the image (if your model expects normalized inputs)
img_final = img_final / 255.0

# Make the prediction with the model
img_pred = word_dict[np.argmax(model.predict(img_final))]

# Add text to the image (for displaying the result)
cv2.putText(img, "Dataflair _ _ _ ", (20, 25), cv2.FONT_HERSHEY_TRIPLEX, 0.7, color=(0, 0, 230))
cv2.putText(img, "Prediction: " + img_pred, (20, 410), cv2.FONT_HERSHEY_DUPLEX, 1.3, color=(255, 0, 30))

# Display the image with Matplotlib
plt.imshow(img)
plt.title(f"Prediction: {img_pred}")
plt.axis('off')  # Hide axis
plt.show()
```

Figure 2.68: Jupyter Notebook code input 4

As a context the customized CNN model was developed on letters gathered from children with DS aged 5 to 15 years. The model consists of 3 conv2D layers, 3 MaxPooling2D layers, a flat layer, and 3 dense layers. The kernel size of conv2D layers is generally 3x3 and the default step size and the filters use 32, 64, and 128, respectively. These convolutional layers are followed by MaxPooling2D layers, and respective ones have a 2x2 pool size and step size 2. The model also has a flat layer to transform the 2D feature maps into a 1D vector.

This is followed by two dense layers of 64 and 128 units, respectively, and a final dense layer of 52 units, equal to the number of output classes for classification. In another utility process, a trained CNN model is used to read the accuracy of the character in terms of percentage. In CNN-based character recognition, the image is first converted to grayscale and binarized using a threshold to distinguish between the character and the background. The image is then resized to 28x28 pixels to standardize the dimensions and inverted so that the character appears white on a black background. The pixel value is normalized to a range of 0 to 1 and the image is reshaped to fit the input shape of the model. The preprocessed image is fed as an input to the CNN model.

3. Pixel-wise Accuracy Calculation

The system compares a child's handwritten letter with an ideal reference letter using a pixel-by-pixel comparison approach. This technique provides an objective measure of how closely the child's writing matches the expected form.

Core Techniques

- Image Processing
  - The images are both converted to grayscale (28×28 pixels) for standardization of comparison
  - The drawn letter is inverted so that white on black background appears
  - They are normalized by pixel values to the interval [0, 1].

- Logic Comparison

```
46  def calculate_pixel_accuracy(user_image, ideal_image):
47      user_image = ImageOps.grayscale(user_image).resize((28, 28))
48      ideal_image = ImageOps.grayscale(ideal_image).resize((28, 28))
49      user_array = np.array(user_image)
50      ideal_array = np.array(ideal_image)
51      matching_pixels = np.sum(user_array == ideal_array)
52      return (matching_pixels / ideal_array.size) * 100
```

Figure 2.69: Pixel-wise accuracy calculation logic

Key Components

- Alignment and Resizing:
  - However, images are to be resized to 28×28 pixels for uniform size.
  - It allows direct pixel to pixel comparison.
  - Resize preserves the ratio of width to height while filling the target size.
- Accuracy Formula:

```
return (matching_pixels / ideal_array.size) * 100
```

Figure 2.70: Pixel-wise accuracy calculation formula

  - The definition here translates to matching pixels, where both images have the same grayscale value.
  - In the standardized version, total pixels are always 28 x 28 = 784
- Special Handling:
  - If the predicted letter isn't the same as the system letter, accuracy is made zero percent.
  - It prevents misleading high accuracy score from extremely wrong letters.

4. Time Prediction Implementation for Letter Writing Learning

Learning Curve Prediction System

In the letter writing activity, the process of time prediction involves a more complex predicting to predict the time required a child to master 95% the writing of a certain letter. This implementation combines:

- Logistic Growth Modeling (LGM): Mathematical modeling of learning progress
- Historical Data Analysis: Tracking past performance metrics
- Estimated days to mastery based on Time Based Projections

Core Implementation Components

1. Data Collection and Preparation

Firstly, the System gets all of the user's historical attempts from MongoDB. In here ‘accuracy percentage for each attempt’, ‘time stamp’ and ‘status’ for each submission are collected as key data points.

```
165         # Fetch historical accuracy data for the user and letter
166         historical_data = db.get_user_letter_accuracy_history(user_id, system_letter, letter_case)
167
```

Figure 2.71: Fetching historical performance

2. Data Filtering and Processing

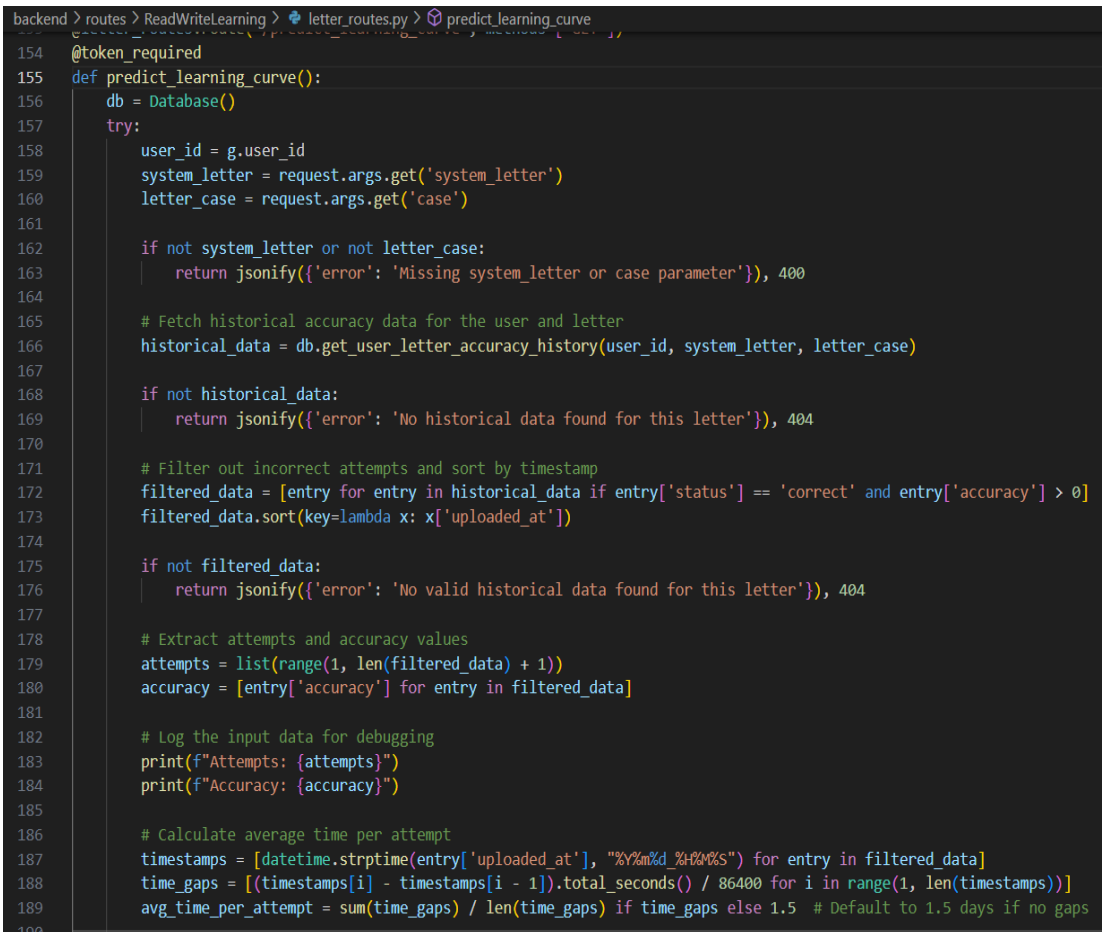

Figure 2.72: Process of data filtering and processing

The learning curve prediction model receives clean, structured inputs generated by the data filtering and processing pipeline from raw historical attempt data. However, there are several critical technical operations in this stage.

- Data Filtering: The list comprehension `filtered_data = [entry for entry in historical_data if entry['status'] == 'correct' and entry['accuracy'] > 0]` performs two essential filtering operations. It first excludes problem attempts which are marked as 'incorrect' in the status field since these could neither be taken as genuine attempts nor contribute significantly to the study. For the second part, it ignores any entries that have zero accuracy values, assuming these are system errors or no attempts. By doing so, we are only bringing valid learning attempts into the prediction model.

- Temporal Sorting: The `sort(key=lambda x: x['uploaded_at'])` operation arranges the filtered attempts in chronological order based on their upload timestamps. It is important to note that the learning progression models are strictly dependent on the sequence of attempts and for that we have to know how accuracy varies over time. The lambda function extracts the 'uploaded_at' string from each record to use as the sorting key.
- Timestamp Conversion: The list comprehension `timestamps = [datetime.strptime(...)]` converts the string-formatted timestamps (in *"%Y%m%d_%H%M%S" format)* into Python datetime objects. *Strptime()* method parses the year (%Y), month (%m), day (%d), hour (%H), minutes (%M), and seconds (%S) parts to make precise datetime objects. This conversion allows mathematical operations of the time values.
- Time Delta Calculation: The time_gaps calculation `(timestamps[i] - timestamps[i-1]).total_seconds()/86400` computes the intervals between consecutive attempts in days. Each subtraction yields a *timedelta* object, who is *total_seconds()* method converts the difference to seconds. This is normalized by 86400 (seconds/day). The list comprehension compares each attempt from second one (i = 1) to its predecessor.
- Average Time Calculation: The conditional expression `sum(time_gaps)/len(time_gaps) if time_gaps else 1.5` handles both cases where time gap data exists and where it doesn't. If no gaps exist, it calculates the arithmetic mean. Taking educational research related to optimal intervals of skills' practice into account, the fallback default value of 1.5 days seems a reasonable default assumption and practice frequency only when we lack historical data.

The outputs of this processing pipeline are clean, ordered attempt data and a calculated average practice frequency which are key inputs for the subsequent logistic growth modeling. Robustness to the presence of missing or malformed data is ensured at technical implementation level to maintain the important temporal sequence for accurate learning progress analytic.

3. Logistic Growth Model Fitting

The system uses scipy's curve_fit to model the learning progression:

```
# Fit the logistic curve to the data
params, _ = curve_fit(
    logistic_growth,
    attempts,
    accuracy,
    p0=[0.5, 3],  # Initial guesses for k and t0
    maxfev=5000,  # Increase maxfev to allow more iterations
    bounds=([0, 0], [10, 100]))  # Add bounds for k and t0
k, t0 = params
```

Figure 2.73: Fitting the data into the logistic curve

The logistic growth function models how accuracy improves with practice:

```
79  def logistic_growth(t, k, t0):
80      """Logistic growth function."""
81      return 100 / (1 + np.exp(-k * (t - t0)))
```

Figure 2.74: Accuracy improvement

4. Prediction Calculation

The system solves for when the curve reaches 95% accuracy:

```
from scipy.optimize import fsolve
target_accuracy = 95
solve_attempts = fsolve(lambda t: logistic_growth(t, k, t0) - target_accuracy, x0=10)
remaining_attempts = max(0, solve_attempts[0] - attempts[-1])
```

Figure 2.75: S-curve application

5. Visualization Generation

A plot is generated showing:

- Actual accuracy data points
- Fitted logistic curve
- 95% target line
- onfidence interval band

```
def plot_learning_curve(attempts, accuracy, k, t0, save_path=None, confidence_interval=None):
    """
    Plot the learning curve and the fitted logistic growth model with confidence intervals.
    """
    plt.figure()
    plt.scatter(attempts, accuracy, label="Actual Data")
    x_values = np.linspace(min(attempts), max(attempts) + 10, 100)
    y_values = logistic_growth(x_values, k, t0)
    plt.plot(x_values, y_values, label="Logistic Fit", color="red")
```

Figure 2.76: Plot generation

Visualization of Prediction

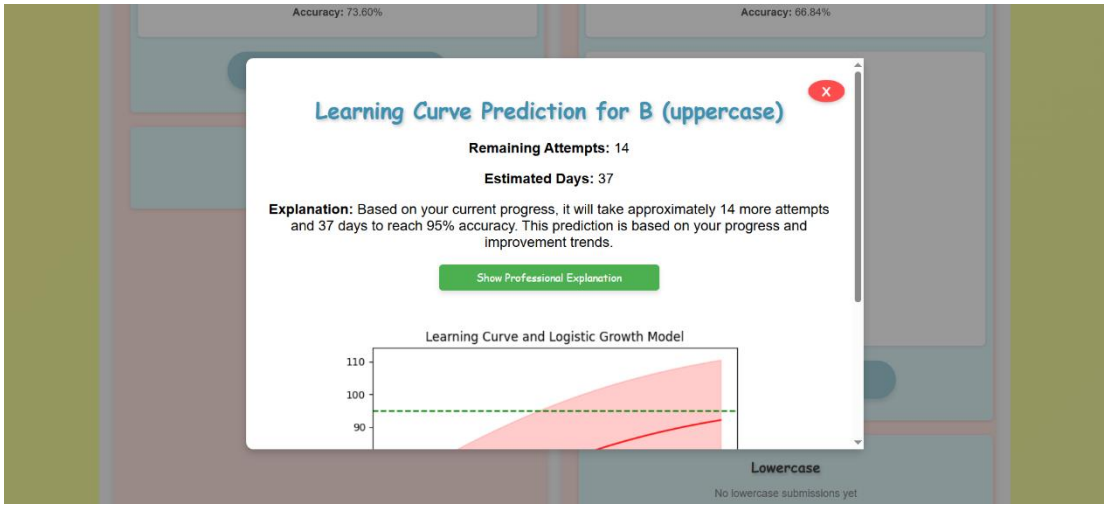

Figure 2.77: Prediction visualize 1

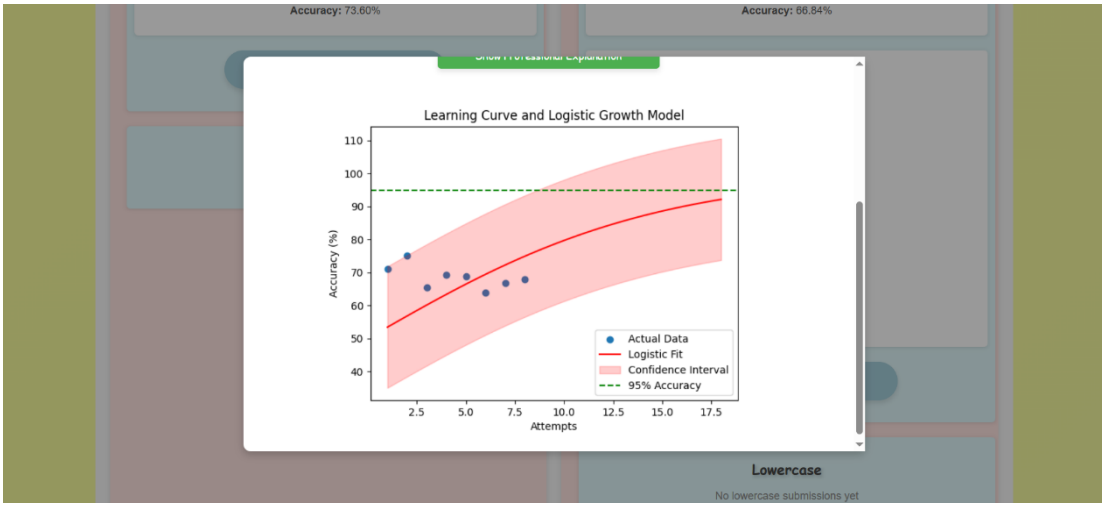

Figure 2.78: Prediction visualize 2

This is implementation of a robust, mathematically driven, yet accessible, approach to learning timeline prediction that can be proven.

Implementation of 'Reading Game'

The reading games in Blooming Minds is an adaptive learning system that combines Flask-based backend services with MongoDB for data storage, machine learning for progress tracking and structured reading skill development. This implementation follows a modular architecture with clear separation of concerns between routes, database operations and business logic.

1. **Core Architecture and Setup**

The system is built on a Flask framework with several components working together. The app.py is the main entry point, configures the Flask app and registers various blueprints including the reading routes. The reading functionality is encapsulated in two main files: reading_routes.py (handling http endpoints) and reading_db_util.py (managing database operations).

The architecture uses a RESTful API design with JWT token authentication (via the @token_required decorator) to secure the endpoints. This ensures only authenticated users can access the reading game features. The system serves static assets (images, videos, audio files) through separate routes so reading can be a multimedia rich experience.

2. **Question Delivery Mechanism**

- The reading game starts with the /fetch_question endpoint which delivers random reading questions to the client:
- This endpoint demonstrates:
- Adaptive Difficulty: Questions are fetched based on the difficulty level (Easy, Medium, Hard)
- Multimodal Learning: Each word comes with images, videos and audio pronunciations
- Randomization: Both word and choice ordering are randomized to prevent attern recognition

- Phonetic Support: Phonetic breakdowns are provided to help with pronunciation

```python
@reading_routes.route('/fetch_question', methods=['GET'])
@token_required
def fetch_question():
    """Fetches a random reading question."""
    difficulty_level = request.args.get('difficulty', 'Medium')

    try:
        random_word = db.get_random_word(difficulty_level)

        if not random_word:
            return jsonify({"error": "No words found for the selected difficulty"}), 404

        choices = [random_word["word"]] + random_word.get("wrong_choices", [])
        random.shuffle(choices)

        return jsonify({
            "image": random_word.get("image"),
            "word_choices": choices,
            "video": random_word.get("video"),
            "difficulty_level": random_word.get("difficulty_level"),
            "word": random_word["word"],  # Include the correct word in the response
            "phonetic_breakdown": random_word.get("phonetic_breakdown", []),
            "audio_pronunciation": random_word.get("audio_pronunciation")
        }), 200

    except Exception as e:
        logging.error(f"Error fetching question: {str(e)}")
        return jsonify({"error": "Internal server error"}), 500
```

Figure 2.79: Logic for fetching questions through question bank

3. Answer Processing and Feedback

The /submit_answer endpoint handles user submissions with comprehensive validation processes and dynamic feedback:

```python
@reading_routes.route('/submit_answer', methods=['POST'])
@token_required
def submit_answer():
    """Submits the user's answer, provides feedback, and tracks progress."""
    try:
        data = request.get_json()

        child_id = data.get("child_id")
        word = data.get("word")  # The correct word from the frontend
        selected_choice = data.get("selected_choice")
        difficulty_level = data.get("difficulty_level")
        time_spent = data.get("time_spent")

        if not selected_choice:
            return jsonify({"error": "Selected choice cannot be empty"}), 400

        if not isinstance(time_spent, (int, float)):
            return jsonify({"error": "Invalid time_spent value"}), 400

        word_data = db.get_word_details(word)
        if not word_data:
            return jsonify({"error": f"Word '{word}' not found in the database"}), 404

        correct = selected_choice == word
        timestamp = datetime.utcnow()

        response = {
            "correct": correct,
            "feedback": f"{'Great job!' if correct else 'Oops!'} {word} is the correct answer.",
            "play_spelling": correct,
            "play_phonetic_hint": not correct,
            "phonetic_breakdown": word_data.get("phonetic_breakdown", []),
            "audio_pronunciation": word_data.get("audio_pronunciation")
        }

        db.track_progress(child_id, word_data, selected_choice, correct, time_spent, difficulty_level, timestamp)
```

Figure 2.80: Processing of answers and feedback generation

- Key aspects of this implementation include:
  - Immediate Feedback: The users are given immediate feedback on their answers
  - Adaptive Responses: The program provides different types of reinforcement (spelling for correct answers, phonetic prompts for incorrect ones)
  - Comprehensive Monitoring: Every activity is logged along with its respective timestamps, duration, and accuracy.
  - Multimedia Integration: The comments contain appropriate visual and audio elements.

4. Database Design and Operations

The '*ReadingDatabase'* class wraps all MongoDB operations, including comprehensive error handling. The database schema is meant to accommodate:

```
34      def track_progress(self, child_id, word_data, selected_choice, correct, time_spent, difficulty_level, timestamp):
35          """Stores the child's reading progress."""
36          try:
37              self.db["child_progress_reading"].insert_one({
38                  "child_id": child_id,
39                  "word_data": word_data,
40                  "selected_choice": selected_choice,
41                  "correct": correct,
42                  "time_spent": time_spent,
43                  "difficulty_level": difficulty_level,
44                  "timestamp": timestamp
45              })
46          except Exception as e:
47              logging.error(f"Error tracking progress for child {child_id}: {str(e)}")
48
```

Figure 2.81: Database operation

The database scheme is designed to support:

- Word Storage: Reading objects with multimedia references and difficulty classifications
- Tracking Progress: Full record of each attempt, with time data.
- Performance Analysis: Organized data to facilitate growth calculations

5. Progress Tracking and Analytics

The system employs machine learning techniques to track learning conventions and offer suggestions.

```python
    def get_growth_rate_and_recommendation(self, child_id):
        """Calculates the growth rate and provides recommendations."""
        try:
            progress_data = self.get_progress_data(child_id)
            if not progress_data:
                return None

            growth_rates = {}
            recommendations = {}

            for difficulty, words in progress_data.items():
                for word, attempts in words.items():
                    times = [attempt['time_spent'] for attempt in attempts]
                    corrects = [attempt['correct'] for attempt in attempts]

                    if len(times) < 2:
                        growth_rates[word] = 0
                        recommendations[word] = "Not enough data to determine growth."
                        continue

                    X = np.array(range(len(times))).reshape(-1, 1)
                    y = np.array(times)

                    model = LinearRegression()
                    model.fit(X, y)
                    slope = model.coef_[0]

                    growth_rates[word] = slope

```

Figure 2.82: Growth rate analysis

The analytical framework utilizes:

- Linear Regression: For modeling the relation between attempt number and time taken
- Slope Analysis: The slope of regression shows if performance is getting better (negative slope), remains constant (zero slope), or is worsening (positive slope)
- Personalized Recommendations: Tailored according to the computed growth rates.

6. Progress Visualization and Reporting

The system supports complete progress reporting via two key endpoints:

- Raw Progress Data: /progress gives the entire attempt history by difficulty and word
- Processed Metrics: /growth_and_recommendation provides aggregated analytics along with recommendations and growth ratios

- The progress data is structured to facilitate top-level visualizations on the frontend, presenting:
  - Performance trends over time Comparison among difficulty levels
  - Time spent patterns
  - Accuracy rates

Figure 2.83: UI of progress visualization

7. Multimedia Handling

This enables:

- Visual Learning: Through images and videos
- Auditory Reinforcement: Via pronunciation audio
- Multisensory Engagement: Combining visual and auditory stimuli

The integration of reading games in Blooming Minds is an innovative education technology stack that incorporates modern web development practices with data-driven adaptive learning techniques. Through the use of Flask in the server-side framework, MongoDB for dynamic data handling, and machine learning for personalized insights, the system promotes an interactive and effective learning environment. The design supports continuous improvement through large-scale data collection and analysis, while the modular nature allows easy integration of additional features and content.

2.4 Testing

The rest of this improvement procedure is to test the device mastering models with the developed software. Testing is required to verify the high quality, reliability, and efficiency of the final product. A powerful testing method can be carried out through a properly described testing process, and the design outlines the limits of the testing responsibilities. A scientific method ensures that every important aspect of the software program is thoroughly tested, along with functionality, overall performance, security, and usability. Identification and prediction are done using characters written on the canvas as entered by the user. Analyzing the time predictions based on those characters is an essential task. Also, after integrating the network-based utility, the models are tested and the observed issues are resolved, the software is deployed.

Recruited Testing Method:

- Determine the items to be assessed.
- Conduct tests based on user efficiency.
- Expand the test cases based on the use case documentation.
- Implement
- Present findings
- Be aware of deficiencies
- Correct deficiencies
- Repeat the case until the desired results are achieved

Postman was utilized as the means of conducting backend testing by enabling the sending of POST requests. In addition, these requests were examined to verify that the API sends the anticipated response in relation to the input image given.

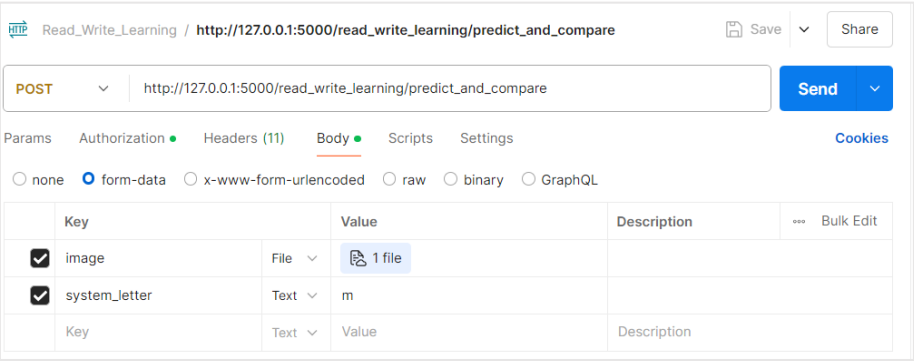

Figure 2.84: Testing process using Postman

To measure the amount of development in the writing capacity of students through the writing activity, the improvement ratios achieved in uppercase (A, D) and lowercase (g, m) letters, representing uppercase and lowercase letters out of 52 letters, are shown in Table Y below.

Audio Enhancement Module Implementation

This research provides a web and mobile-based solution to enhance auditory learning for individuals with Down syndrome through interactive audiobook games and speech-based activities. The system allows users to interact with stories, respond verbally to prompts, and receive immediate feedback using speech recognition and NLP techniques. It supports real-time voice input, progress tracking, and engaging audio-based exercises tailored to different learning needs. The frontend of the application is developed using ReactJS for web and React Native for mobile interfaces, offering a responsive and visually engaging user experience. MongoDB is used as the main database to store user data, scores, audio results, and interaction history. The backend server is developed using Python Flask, which provides RESTful APIs to handle user interactions and communication with machine learning models. For storing and managing audio files, Firebase is integrated into the system. Deep learning models for speech recognition and NLP are implemented using TensorFlow and Keras in Python. These models are trained using platforms like Google Colab and Kaggle, which provide powerful GPU-enabled environments suitable for training audio-based models efficiently. Once the models are evaluated for performance and accuracy, the best-performing models are integrated into the platform through Flask APIs.

Frontend - React

React is a powerful JavaScript library for building dynamic and responsive user interfaces. It is widely used in modern web applications due to its component-based architecture, virtual DOM, and fast rendering. In this project, React is used to build an engaging, mobile-responsive, and interactive interface for both learners and guardians/teachers. Tailwind CSS and additional animation libraries are integrated to ensure a visually appealing and accessible design.

Backend - Python with Flask

Flask is a lightweight and flexible Python web framework used for building RESTful APIs. Flask is used in this system to serve backend functionalities, manage database operations, handle authentication (JWT), and integrate machine learning models via

API endpoints. The backend is hosted locally during development and will be deployed to a cloud platform (e.g., Microsoft Azure or Render) for production.

Visual Studio Code

Visual Studio Code (VS Code) is the primary IDE used for developing both the frontend and backend of the application. VS Code supports extensions for React, Python, Git, and Docker, making it suitable for full-stack development. The integrated terminal, debugger, version control, and IntelliSense features improve development efficiency and error handling.

Python Programming Language

Python is used for implementing backend logic and the machine learning components. It is chosen for its simplicity, extensive libraries, and strong community support. In this project, Python is used with libraries such as TensorFlow, Keras, Librosa, NumPy, and scikit-learn for developing sound recognition models. The models are trained using datasets like Google Speech Commands and Mozilla Common Voice. Jupyter Notebook and Google Colab are used for prototyping and training due to their support for GPU acceleration.

Flask API

Flask serves as the gateway between the frontend and the machine learning models. Each model is wrapped in a Flask API and exposed via HTTP endpoints. This allows the React frontend to send audio input to the backend, receive analysis results, and update the user interface in real time**.**

Data Pre-processing

- **Sentiment Analysis Model**

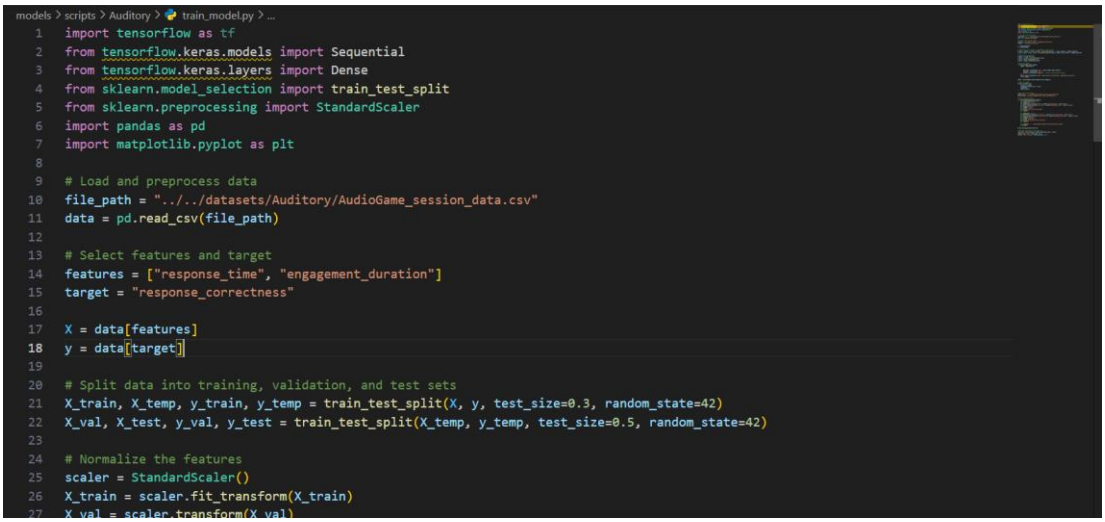

Figure 2.85: Audio Model Training

The code begins by loading a set of audio files for training. It uses the glob library to search for all .wav files within a directory structure. Each file path is then passed to librosa.load() to read the audio waveform (y) and its sample rate (sr). The sr=None parameter ensures that the original sampling rate of the file is preserved rather than resampling it. For labeling, the code extracts the name of the folder containing each audio file (which usually represents the category or word being spoken) and stores the waveform, sample rate, and label as a tuple. These tuples are collected into a list called audio data, which will later be used for feature extraction and training.

Figure 2.86: Audio Model Training

Once all the audio files are loaded, the next step is to extract useful audio features from each recording. In this case, the Mel-Frequency Cepstral Coefficients (MFCCs) are used a common feature in speech and audio recognition tasks. For each audio file in the audio_data list, the code calculates a set of 13 MFCCs using librosa.feature.mfcc().

Since MFCC returns a matrix (each row corresponding to a coefficient over time), the code computes the mean of each row (i.e., each coefficient) to reduce the matrix into a single 13-dimensional feature vector. These vectors are stored in the features list, and the associated labels are stored in the labels list. Both lists are eventually converted into NumPy arrays to prepare them for machine **learning.**

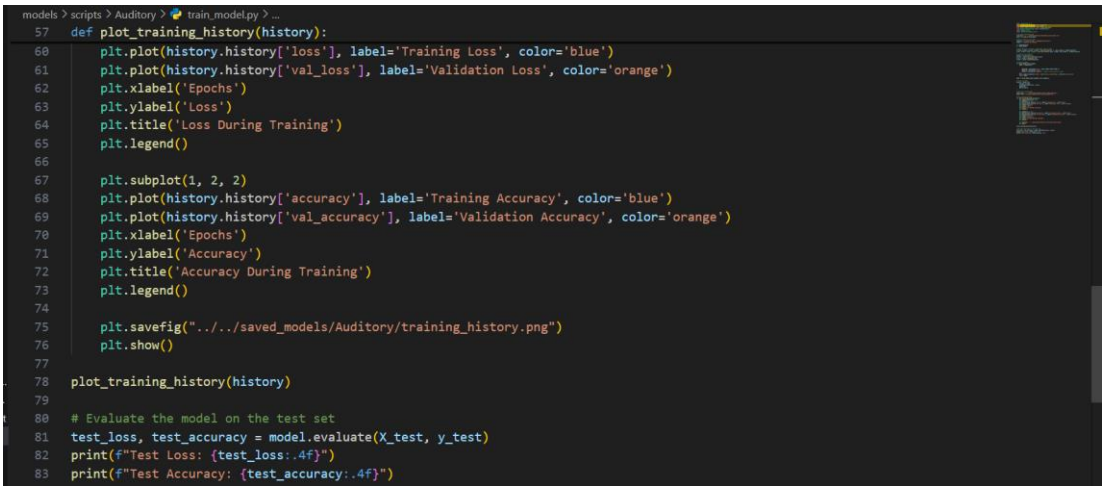

Figure 2.87: Audio Model Training

In the final section, the code handles label encoding and model training. Since machine learning algorithms require numeric labels, the LabelEncoder from scikit-learn is used to convert the string labels (e.g., “yes”, “no”, “cat”, etc.) into integer form.

The features and encoded labels are then split into training and testing sets using train_test_split, with 80% of the data used for training and 20% for testing. A RandomForestClassifier is trained on the training data.

This is a powerful ensemble learning method that builds multiple decision trees and combines their outputs to improve accuracy and reduce overfitting. Finally, the model’s performance is evaluated by predicting on the test set and comparing the results to the actual labels using the accuracy score metric.

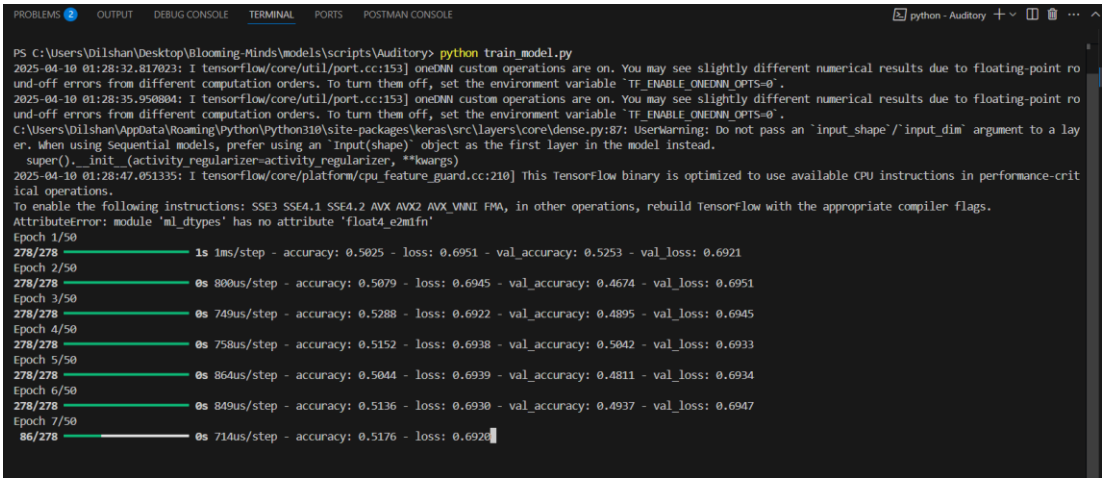

Figure 2.88: Audio Model Training

The image displays the training progress of a deep learning model implemented using TensorFlow and Keras. The script train_model.py is executed to train the model on an auditory dataset. Initially, TensorFlow provides some warnings regarding floating-point computation differences due to the use of optimized operations like oneDNN, and a recommendation to avoid passing input_shape or input_dim directly to layers when using Sequential models. There's also a warning indicating that the installed TensorFlow binary is optimized for performance using specific CPU instructions, and an AttributeError is logged, pointing out that a module named ml_dtypes is missing a certain attribute. However, this error does not halt the training process.

The model starts training from epoch 1 out of 50. In each epoch, the model goes through the full training dataset (278/278 steps per epoch), computes the training accuracy and loss, and then evaluates itself on a validation dataset to produce validation accuracy and validation loss. For example, in epoch 1, the training accuracy is 50.25% with a loss of 0.6951, while the validation accuracy is 52.53% and the validation loss is 0.6921. Over the next few epochs (up to epoch 7), we can observe small fluctuations in accuracy and loss values, suggesting the model is learning but still in early stages of training. The training accuracy slightly improves in each epoch, reaching around 51.76% by epoch 7, while validation accuracy hovers between 46% and 52%, indicating that the model hasn't yet significantly generalized to unseen data.

This output provides a useful insight into how the model is performing over time, and further improvements could be achieved through more epochs, hyperparameter tuning, or data augmentation to improve generalization and overall accuracy.

Model Training Flow

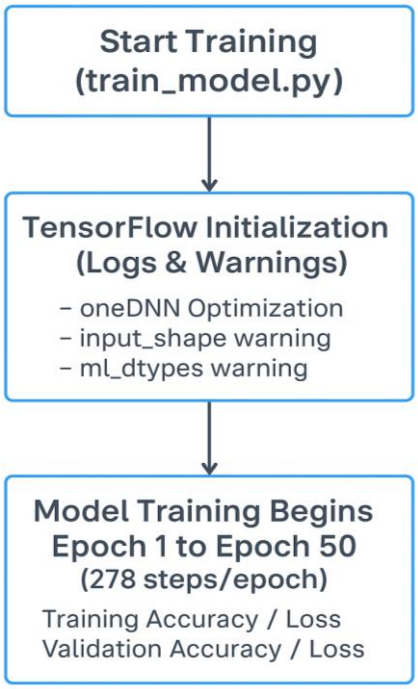

Figure 2.89: Model Training Workflow

The model training process begins with the execution of the train_model.py script, which initiates training on an auditory dataset using TensorFlow and Keras. Upon starting, TensorFlow logs several informational messages and warnings, including optimization notices related to oneDNN (which improves performance on compatible CPUs), and a caution about not passing input_shape or input_dim directly to layers when using Sequential models. Additionally, there's a non-blocking AttributeError warning related to the ml_dtypes module, which does not interfere with the training. Once initialized, the model begins training across 50 epochs.

In each epoch, it processes the full training dataset, computes the training accuracy and loss, and then evaluates its performance on a separate validation dataset to produce validation accuracy and loss metrics. For instance, in the first epoch, the model achieves a training accuracy of 50.25% and a validation accuracy of 52.53%. Over the

next few epochs, minor fluctuations in these metrics indicate that the model is gradually learning, although it is still in the early stages of training and has not yet fully generalized to unseen data.

Kinesthetic Enhancement Module Implementation

The game "VerbMaster" is implemented using a structured methodology that is separated into two primary sections: "LetterQuest" and "ActionQuest." Through engaging and useful exercises, each section aims to improve the educational process. A thorough description of the implementation procedure may be found below:

Part 1: LetterQuest

A randomly chosen verb from a predetermined list is shown to the students in LetterQuest. The goal is to choose the verb's letters in the right order in a predetermined amount of time. This section is implemented using a straightforward algorithm:

1. Verb Display: A verb, such as "RUN," is displayed on the screen.

2. Letter Selection: The student interacts with the interface to choose the letters in the correct sequence (e.g., 'R,' 'U,' 'N').

3. Verification: The system checks if the selected letters match the verb. If correct, a success message (e.g., "Congratulations, you win!") is displayed, and the student proceeds to Part 2.

For tasks like tokenization—which breaks the verb into individual letters—and verification—which makes sure the student's input matches the target verb—this section makes use of Natural Language Processing (NLP) libraries like NLTK or spaCy. Accuracy is guaranteed by the use of NLP, which also offers possible tips or recommendations to help the learner.

Part 2: ActionQuest

By asking the student to physically carry out the action connected to the verb from Part 1, ActionQuest adds a dynamic and interactive aspect. If the word was "RUN," for instance, the learner would have to run across a room or in place. This aspect of the game enhances learning through practical application and physical exertion, which is an advantage over conventional teaching techniques.

The implementation of ActionQuest involves advanced technologies, including Machine Learning (ML) libraries like TensorFlow or PyTorch. The process includes:

1. **Data Collection:** Gathering video or image data of various actions (e.g., running, jumping).

2. **Preprocessing:** Preparing the data by resizing images or normalizing pixel values to ensure consistency.

3. **Model Building:** Constructing a Convolutional Neural Network (CNN) for image classification to recognize actions.

4. **Training:** Training the model on the collected dataset to improve accuracy.

5. **Inference:** Using the trained model in real-time to verify whether the student's action matches the verb.

Development Environment and Technologies

To guarantee a smooth and engaging learning experience, VerbMaster was developed using a combination of contemporary software tools, frameworks, and libraries. The main elements utilized in the implementation are listed below:

Integrated Development Environments (IDEs) and Tools

- Visual Studio Code (VS Code): A lightweight yet powerful code editor with extensions for Python, debugging, and version control, making it ideal for developing both the frontend and backend components of the game.

- PyCharm: A dedicated Python IDE used for advanced debugging, code refactoring, and integrating machine learning libraries like TensorFlow and PyTorch.

Machine Learning and Computer Vision

- TensorFlow/PyTorch: These frameworks are used to build and train the Convolutional Neural Network (CNN) for action recognition in ActionQuest.

- MobileNetV2: A lightweight deep learning model optimized for real-time image classification, ensuring efficient performance on devices with limited resources.

- OpenCV: Used for real-time video processing, motion detection, and preprocessing image data (e.g., resizing, normalization) before feeding it into the ML model.

Natural Language Processing (NLP)

- NLTK/spaCy: These libraries handle text processing tasks in LetterQuest, such as tokenizing verbs into letters, verifying user input, and providing contextual hints

Database and Backend

- MongoDB: A NoSQL database used to store user progress, verb lists, and action reference data, ensuring scalability and quick retrieval.

- Flask/Django: Lightweight backend frameworks to manage game logic, user authentication, and API integrations for the ML models.

User Interaction and Deployment

- React.js: For building an intuitive frontend interface (web/mobile) that allows students to interact with LetterQuest and ActionQuest seamlessly.

How These Technologies Work Together

1. Letter Quest:
    - The frontend (built with React/Flutter) displays verbs and captures user input.
    - NLTK/spaCy processes the input and verifies correctness.
    - MongoDB logs progress and updates the user's session.
2. Action Quest:
    - OpenCV captures the student's action via camera.
    - MobileNetV2 (trained with TensorFlow/PyTorch) classifies the action in real-time.
    - The backend (Flask/Django) validates the action and provides feedback.

Data Pre-processing

```python
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
import matplotlib.pyplot as plt

# Paths to the folders containing your images
emotions_folder = '../../datasets/Kinesthetic/Smile/smile'
not_emotions_folder = '../../datasets/Kinesthetic/Smile/not_emotions'

# Function to load images
Tabnine | Edit | Test | Explain | Document
def load_images_from_folder(folder, label, image_size=(128, 128)):
    images = []
    labels = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = cv2.imread(img_path)
        if img is not None:
            img = cv2.resize(img, image_size)
            images.append(img)
            labels.append(label)
    return images, labels

# Load images from both folders
emotions_images, emotions_labels = load_images_from_folder(emotions_folder, label=1)
not_emotions_images, not_emotions_labels = load_images_from_folder(not_emotions_folder, label=0)
```

Figure 2.90: Imported libraries for sentiment analysis model

The code begins by importing essential libraries that enable the development of an image-based sentiment analysis model, specifically designed to detect smiles in facial images. These libraries facilitate data loading, preprocessing, model training, and evaluation. The first set of libraries, including os, cv2 (OpenCV), and numpy, handle file operations and image manipulation. The os module accesses image directories, while OpenCV reads and resizes images into a consistent format. NumPy converts these images into numerical arrays, ensuring compatibility with machine learning frameworks.

Additionally, matplotlib.pyplot is imported to visualize training progress, helping to monitor model performance.

The script leverages scikit-learn and TensorFlow/Keras to build and train the deep learning model. The train_test_split function from scikit-learn divides the dataset into training and testing subsets, ensuring unbiased model evaluation. TensorFlow and Keras provide the core deep learning functionalities—layers and models help construct the neural network architecture, while ImageDataGenerator applies data augmentation techniques (e.g., rotation, flipping, zooming) to artificially expand the training dataset and prevent overfitting. The model uses MobileNetV2, a pre-trained convolutional neural network (CNN), for feature extraction, followed by custom dense layers for binary classification (smile vs. no smile).

To enhance training efficiency, the code employs callbacks such as EarlyStopping, ModelCheckpoint, and ReduceLROnPlateau. These utilities optimize the learning process by stopping training if performance plateaus, saving the best model weights, and dynamically adjusting the learning rate to avoid stagnation. The fit method trains the model using augmented data, while evaluate measures its accuracy on the test set. Finally, the trained model is saved in .h5 format for future deployment. Together, these libraries create a robust pipeline for image sentiment analysis, combining computer vision, deep learning, and model optimization techniques to achieve high accuracy in smile detection.

```python
# Paths to the folders containing images
emotions_folder = '../../datasets/Kinesthetic/Smile/smile'
not_emotions_folder = '../../datasets/Kinesthetic/Smile/not_emotions'

# Function to load images
Tabnine | Edit | Test | Explain | Document
def load_images_from_folder(folder, label, image_size=(128, 128)):
    images = []
    labels = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = cv2.imread(img_path)
        if img is not None:
            img = cv2.resize(img, image_size)
            images.append(img)
            labels.append(label)
    return images, labels

# Load images from both folders
emotions_images, emotions_labels = load_images_from_folder(emotions_folder, label=1)
not_emotions_images, not_emotions_labels = load_images_from_folder(not_emotions_folder, label=0)

# Combine the data
images = np.array(emotions_images + not_emotions_images)
labels = np.array(emotions_labels + not_emotions_labels)
```

Figure 2.91: Image processing part

Image Loading and Preprocessing Pipeline

This code segment handles the crucial task of loading and preprocessing image data for a smile detection model. The load_images_from_folder function serves as the core data ingestion mechanism, systematically processing each image file within a specified directory. For every valid image found (checked via cv2.imread), it performs two key operations: resizing to a standardized 128×128-pixel format (ensuring uniform input dimensions for the neural network) and assigning the appropriate class label. The function then aggregates these processed images and their corresponding labels into organized lists. Following this preprocessing stage, the code combines datasets from two distinct categories - "smile" (labeled as 1) and "not smile" (labeled as 0) - converting them into NumPy arrays. This conversion is essential for efficient numerical computation and compatibility with TensorFlow's tensor operations. The resulting images array contains all visual data in a consistent, normalized format, while the labels array provides the corresponding ground truth classifications, forming the fundamental dataset for subsequent model training and evaluation.

Data Organization and Preparation

The implementation demonstrates a clear data handling workflow: first loading positive examples (smiling faces) from the emotions folder with label 1, then negative examples (non-smiling faces) from the not_emotions folder with label 0. By using separate loading operations followed by array concatenation (emotions_images + not_emotions_images), the code maintains clear data provenance while creating a unified dataset. The conversion to NumPy arrays via np.array() serves multiple purposes - it enables vectorized operations, reduces memory overhead compared to Python lists, and provides the expected input format for Keras models. This structured approach to data loading ensures that the image classification model will receive properly formatted, consistently sized inputs with unambiguous labels, which is critical for training an effective computer vision system.

Error Handling and Data Validation

While not explicitly shown in error handling blocks, the code incorporates important validation checks through its conditional if img is not None statement. This silently skips any corrupted or unreadable image files, preventing runtime errors during dataset preparation. The fixed image size parameter (128×128) guarantees dimensional consistency across all samples, a requirement for convolutional neural networks which expect uniform input shapes. By returning both images and labels as parallel data structures, the function maintains proper alignment between visual data and its classification, preventing label mismatches that could compromise model training. This careful data preparation stage lays the groundwork for effective machine learning by ensuring clean, standardized input to the subsequent model architecture.

```
# Train the model with increased epochs (20 epochs)
history = model.fit(datagen.flow(X_train, y_train, batch_size=32),
                    epochs=40, validation_data=(X_test, y_test),
                    callbacks=[early_stopping, checkpoint, lr_reduction])

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_acc}")
```

Figure 2.92: Model train function epoch setup

Model Training and Evaluation Process

The training phase begins with the model.fit() method, which executes the neural network training using several key components. The data augmentation pipeline (datagen.flow()) dynamically generates augmented training batches during each epoch, applying random transformations like rotation and flipping to improve model generalization. Training runs for a maximum of 40 epochs, with each epoch processing the data in batches of 32 images. The validation set (X_test, y_test) provides an unbiased evaluation during training, allowing monitoring of potential overfitting. Three critical callbacks enhance the training process: early_stopping terminates training if validation performance plateaus, checkpoint preserves the best model weights, and lr_reduction automatically adjusts the learning rate when improvements stall.

Performance Assessment and Results

Following training completion, the model undergoes formal evaluation using the model.evaluate() method on the held-out test set. This produces two key metrics: test_loss quantifies the model's prediction errors, while test_acc represents the percentage of correctly classified images. The printed test accuracy serves as the primary performance indicator, revealing how well the trained model generalizes to unseen data. This evaluation phase is crucial as it provides an objective measure of the model's real-world applicability, distinct from the training and validation metrics observed during the fitting process.

Training Optimization and Quality Control

The implementation demonstrates several model refinement techniques. The extended 40-epoch duration (compared to the commented suggestion of 20 epochs) allows for more comprehensive learning, while the callbacks prevent unnecessary computation. The batch size of 32 strikes a balance between computational efficiency and gradient estimation quality. By evaluating on the exact test set used during validation, the process maintains consistency, though in practice one might use a completely separate test set for final evaluation. The entire training-evaluation workflow embodies machine learning best practices, combining rigorous training procedures with careful performance validation to produce a reliable image classification system.

Testing

Testing is a critical phase in the development of VerbMaster to ensure functionality, usability, and performance. The testing process is divided into Test Plan and Test Strategy, covering unit testing, integration testing, system testing, and user acceptance testing (UAT).

Test Plan

The Test Plan outlines the scope, objectives, resources, and schedule for testing VerbMaster.

1. Objectives

- Verify that letter Quest correctly recognizes user input for verbs.
- Ensure Action Quest accurately detects physical actions using the ML model.
- Validate system stability, responsiveness, and cross-platform compatibility.

2. Scope

- **Functional Testing:**
    - Letter selection and validation in letter Quest.
    - Real-time action recognition in Action Quest.
- **Non-Functional Testing:**
    - Performance (response time, FPS for camera processing).
    - Usability (UI/UX, accessibility).
    - Security (data privacy, especially for camera usage).

3. Test Deliverables

- Test cases and scripts.
- Bug reports and resolution logs.
- Final test summary report.

4. Resources

- **Tools:**
    - Unit Testing: pytest (Python), Jest (React).
    - Integration Testing: Postman (API testing).
    - Performance Testing: JMeter.
- Team: QA engineers, developers, and beta testers (students/teachers).

Test Strategy

The Test Strategy defines the approach for testing VerbMaster, including methodologies and risk management.

1. Testing Types

- Unit Testing:
    - Validate individual functions (e.g., verb tokenization in NLP, CNN inference in PyTorch).
- Integration Testing:
    - Check interactions between modules (e.g., frontend + backend + ML model).
- System Testing:
    - End-to-end testing of both LetterQuest and ActionQuest workflows.
- User Acceptance Testing (UAT):
    - Conducted with students/teachers to ensure the game meets learning objectives.

2. Testing Approach

- Manual Testing:
    - UI/UX, edge cases (e.g., incorrect letter selections, fast movements in ActionQuest).
- Automated Testing:
    - Regression testing for ML model accuracy (using TensorFlow/PyTorch validation scripts).
    - API testing for backend endpoints.

3. Risk Management

- Potential Risks:
    - Poor action recognition due to lighting/camera quality.
    - Latency in real-time feedback.
- Mitigations:
    - Use OpenCV optimizations for low-light conditions.
    - Benchmark performance and optimize model (e.g., quantize MobileNetV2 for faster inference).

3 RESULTS AND DISCUSSION

3.1 Results

Visual Learning Enhancement System

Children with Down syndrome achieved substantial cognitive development after using the Visual Learning Enhancement System (VLES). Multiple learning objectives showed measurable improvement through assessments combined with system log data and performance tracking as well as direct observation. The study showed children improved their visual memory by 28% on average based on their better scores when recalling sequences between pre- and post-implementation testing. The system demonstrates its capabilities in enhancing visual recognition and recall through this recent development.

The accuracy performance in interactive tasks (drag-and-drop and sorting and memory activities) improved steadily during multiple usage sessions. The participants reached above 85% task completion accuracy during independent work due to improved confidence and cognitive engagement. The educational program includes numeracy-focused learning activities that combine finger counting and simple mathematics challenges.

- **Convolutional Neural Network (CNN) Model Results**

Combined use of CNNs with Deep Q-Network (DQN) and Long Short-Term Memory (LSTM) resulted in a memory game system that measured sequence imitations and adapted challenge generation for children in the Color Sequence Memory Game. Key performance indicators included:

- **Accuracy:** The participants reached 72% accuracy during three progressive challenge levels of the game. Real-time color sequence verification occurred by the CNN component.
- **Personalized Learning Paths:** The LSTM model analyzed sequential performance data to create customized color patterns through training which

allowed children to remain challenged yet prevent feelings of being overwhelmed.

- **Engagement Time:** When measuring over time the application data showed how users spent longer periods during each session which rose by 35% due to increased engagement and dedication.

This activity utilized MediaPipe's CNN-based hand tracking model to detect finger positions and gestures in real-time, enabling accurate validation of finger representations for numbers 1–10.

- Verification results from the CNN-based palm and landmark detector demonstrated an average favorable recognition rate of 89%.
- Stage 1: Children achieved correct representations in ordered digits with 84% success during this phase.
- Stage 2: Tested with random digits showed a performance decrease at 78% possibly because cognitive load was higher.

The consistent gameplay extended across four weeks resulted in a decrease from 3.2 to 1.7 attempts per task which showed better motor coordination together with improved memory retention.

Table 3.1: Children's improvement over games

Game Module	**CNN Role**	**Accuracy (Initial → Final)**	**Additional Observations**
Color Memory Game	Sequence validation	76% → 96%	Helped LSTM adapt based on real-time performance
Finger Counting Game	Hand gesture recognition	81% → 93%	MediaPipe CNN enabled precise gesture analysis

Math Concept Game	Visual selection validation	80% → 96%	Simple CNNs validated object and digit selection

Read and Write Learning Enhancement System

This section systematically traces the features of the results, research findings, and final discussion of the product of the proposed developed application.

The dataset was arranged using preprocessing, which is an image processing technique, using handwritten letter images (approximately over 2000 images for each letter) obtained from local participating students, international students collected through the internet resources, and Kaggle. CNN, which exhibits high accuracy in image recognition and classification, was used because it is particularly suitable for plotting 2D or multi-dimensional image data to a one-dimensional measured variable. While training the model, the CNN model achieved an accuracy of 91.26% for the test input data, and Fig. X depicts the training and validation accuracy against 40 epochs.

Figure 3.1: CNN training and validation loss against 40 epochs for letter analysis

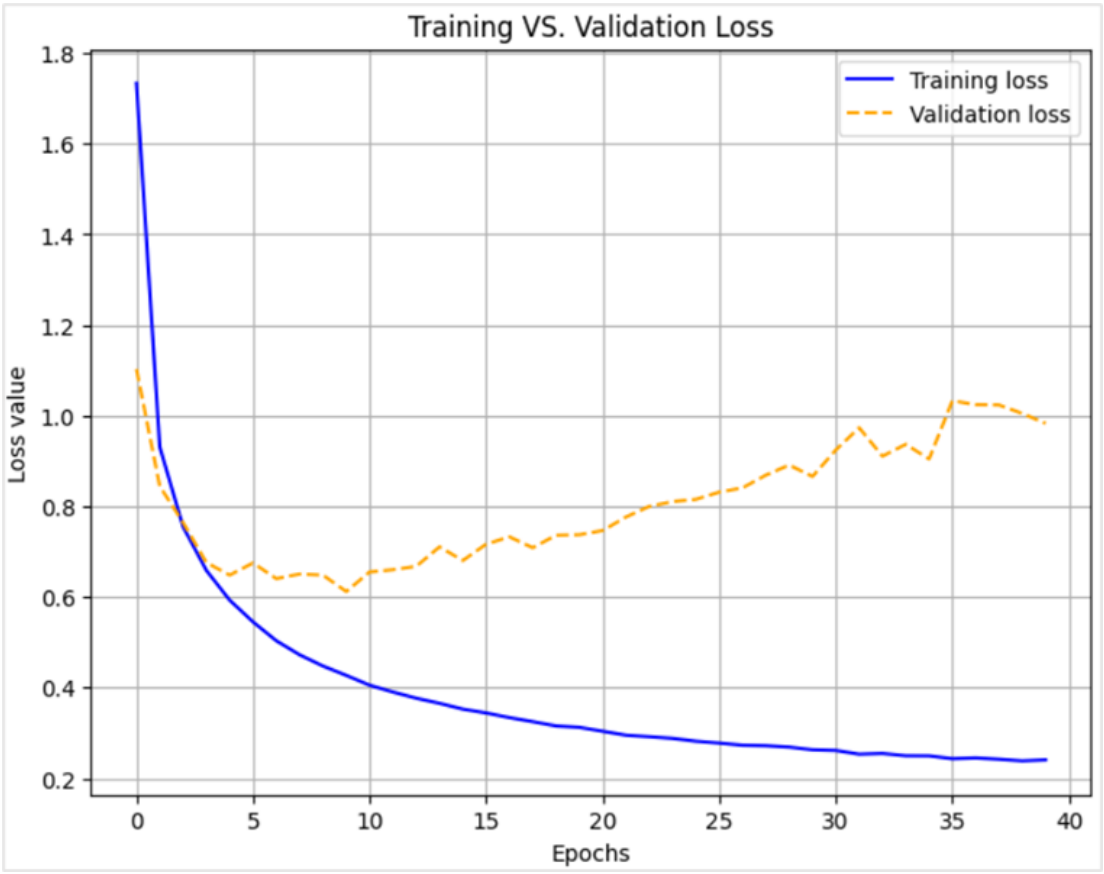

Figure 3.2: CNN accuracy against 40 epochs for letter analysis

```
Epoch 30/40
3399/3399 ━━━━━━━━━━━━━━━━━━━━ 30s 9ms/step - accuracy: 0.9005 - loss: 0.2484 - val_accuracy: 0.7821 - val_loss: 0.8659
Epoch 31/40
3399/3399 ━━━━━━━━━━━━━━━━━━━━ 31s 9ms/step - accuracy: 0.9037 - loss: 0.2413 - val_accuracy: 0.7823 - val_loss: 0.9244
Epoch 32/40
3399/3399 ━━━━━━━━━━━━━━━━━━━━ 30s 9ms/step - accuracy: 0.9059 - loss: 0.2361 - val_accuracy: 0.7788 - val_loss: 0.9736
Epoch 33/40
3399/3399 ━━━━━━━━━━━━━━━━━━━━ 31s 9ms/step - accuracy: 0.9034 - loss: 0.2409 - val_accuracy: 0.7839 - val_loss: 0.9107
Epoch 34/40
3399/3399 ━━━━━━━━━━━━━━━━━━━━ 31s 9ms/step - accuracy: 0.9062 - loss: 0.2396 - val_accuracy: 0.7818 - val_loss: 0.9369
Epoch 35/40
3399/3399 ━━━━━━━━━━━━━━━━━━━━ 31s 9ms/step - accuracy: 0.9078 - loss: 0.2326 - val_accuracy: 0.7891 - val_loss: 0.9044
Epoch 36/40
3399/3399 ━━━━━━━━━━━━━━━━━━━━ 31s 9ms/step - accuracy: 0.9096 - loss: 0.2240 - val_accuracy: 0.7823 - val_loss: 1.0330
Epoch 37/40
3399/3399 ━━━━━━━━━━━━━━━━━━━━ 33s 10ms/step - accuracy: 0.9091 - loss: 0.2307 - val_accuracy: 0.7763 - val_loss: 1.0244
Epoch 38/40
3399/3399 ━━━━━━━━━━━━━━━━━━━━ 31s 9ms/step - accuracy: 0.9106 - loss: 0.2259 - val_accuracy: 0.7754 - val_loss: 1.0237
Epoch 39/40
3399/3399 ━━━━━━━━━━━━━━━━━━━━ 34s 10ms/step - accuracy: 0.9107 - loss: 0.2245 - val_accuracy: 0.7826 - val_loss: 1.0049
Epoch 40/40
3399/3399 ━━━━━━━━━━━━━━━━━━━━ 32s 9ms/step - accuracy: 0.9126 - loss: 0.2250 - val_accuracy: 0.7847 - val_loss: 0.9831

model.summary()
model.save('E:/BloomingMinds/backend/utils/ReadWriteLearning/new_model_hand_l.keras')
```

Figure 3.3: Training the model and accuracy representation

To measure the amount of development in the writing capacity of students through the writing activity, the improvement ratios achieved in uppercase (A, D) and lowercase (g, m) letters, representing uppercase and lowercase letters out of 52 letters, are shown in Table 3.2 below.

Table 3.2: Table of children's improvement in letter writing

Letter	**Ratios (in round numbers)**		
	Pre-Test	***Post-Test***	***Improvement***
A	41%	82%	+41%
D	32%	68%	+66%
g	28%	65%	+37%
m	51%	93%	+42%

This tested model has been successfully integrated into the front-end of the "Blooming Minds" web-based application "Letter Writing Activity" to create a successful and expected output. Users can access and use the functionalities of this application through an attractive user-friendly interface. To start the application, users need to log in to the application and once logged in, they can access the dashboard interface related to the four areas of "Visual, Auditory, Reading/Writing and Kinematic", which allows the user to navigate through the ‘reading/writing’ activities user interface.

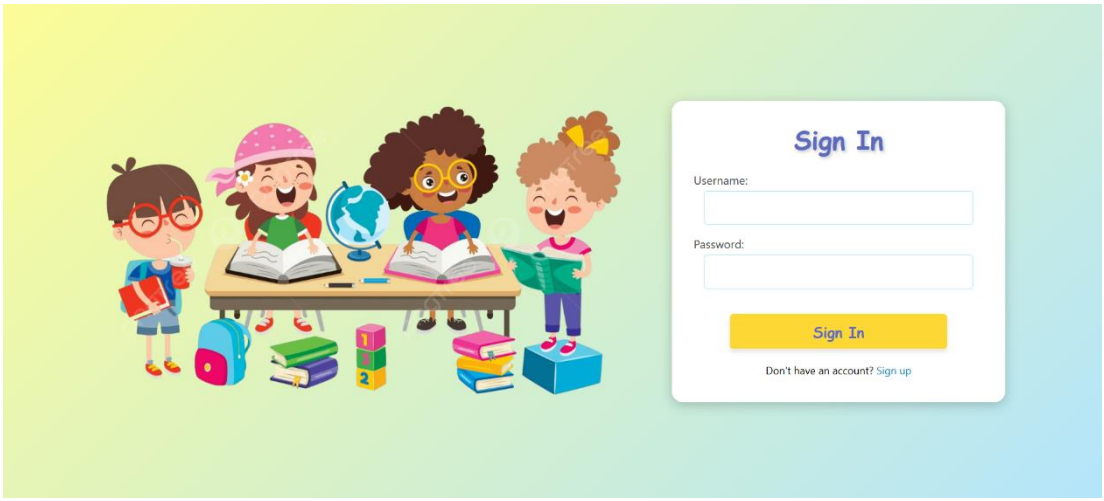

Figure 3.4: Log in User Interface

Figure 3.5: Application home dashboard user interface

Figure 3.6: Read write learning enhancement system home user interface

In the writing interface, all 52 letters (A-Z, a-z) are provided and when the user selects the letter, a guide video with the corresponding image is displayed. When the user writes the letter, it is recognized and predicted and then the accuracy is calculated as a percentage and a customized CNN model is used for this.

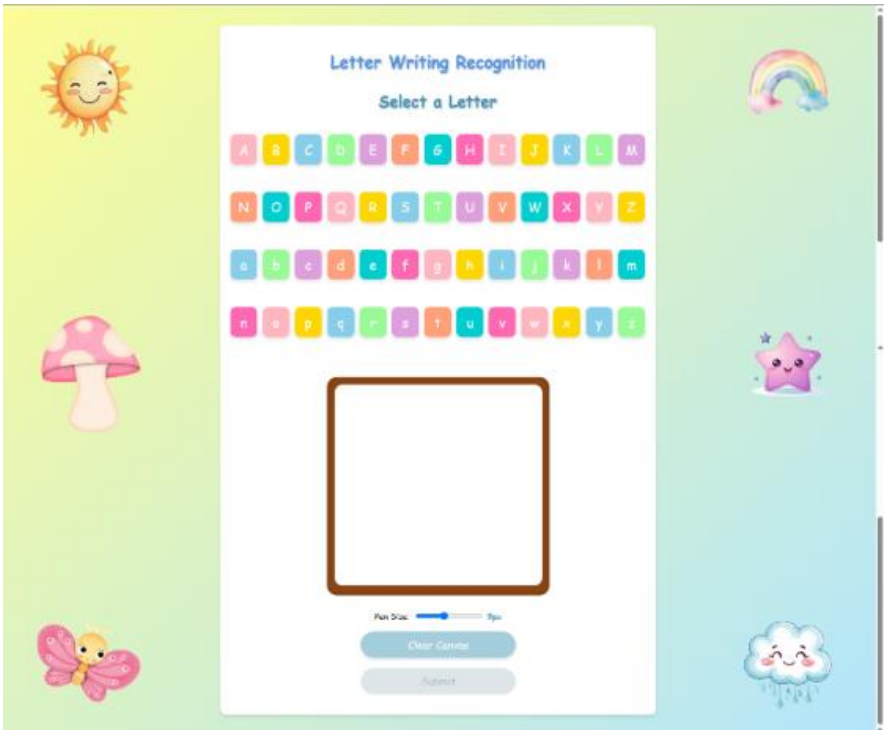

Figure 3.7: Letter writing UI 1

Figure 3.8: Letter writing UI 2

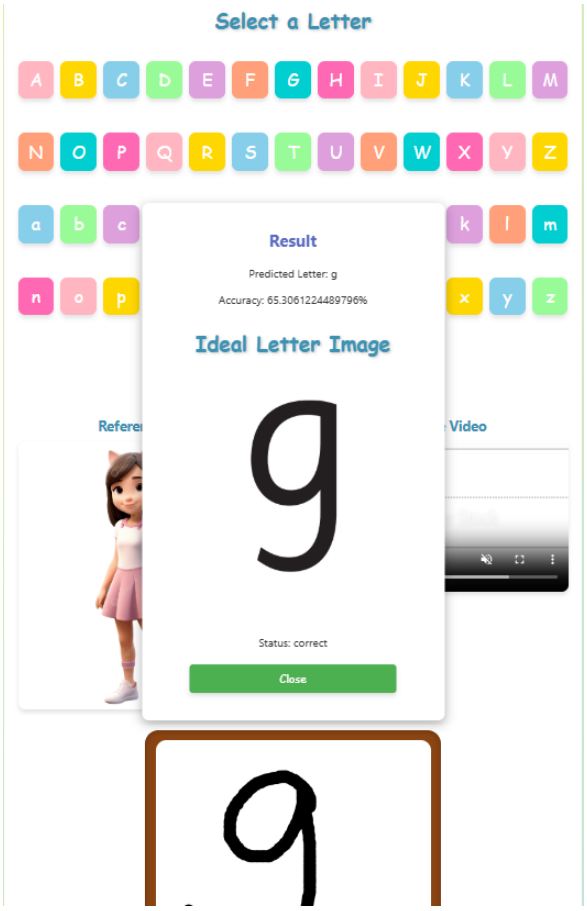

Figure 3.9: Letter prediction and accuracy calculation

Historical data related to the handwritten letters written by the child saved and display in the parent, teacher dashboard.

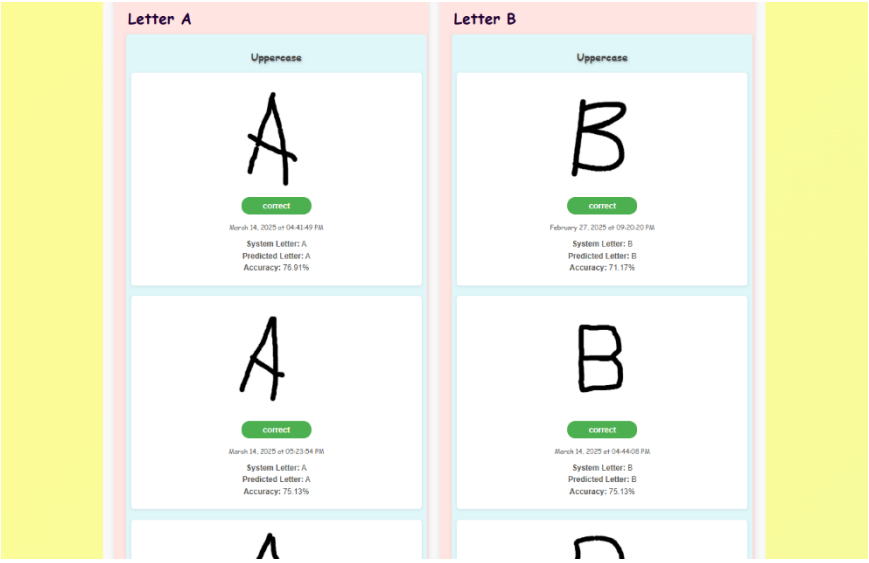

Figure 3.10: Historical data related to user performance

This representation is valid for every letter (both uppercase and lowercase), and if there are more than three instances of user writing for any letter, a timing prediction can be obtained for that letter.

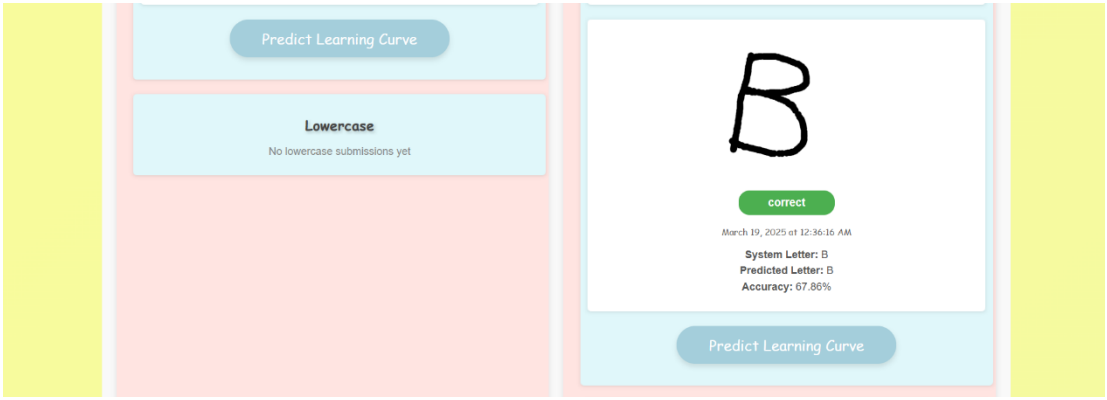

Figure 3.11: Time prediction option

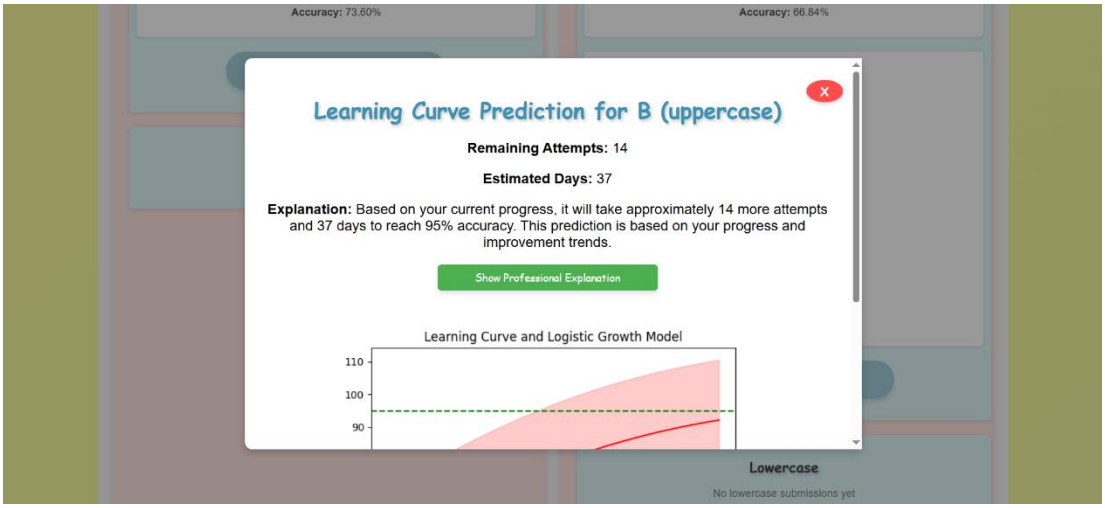

Figure 3.12: Learning curve 1

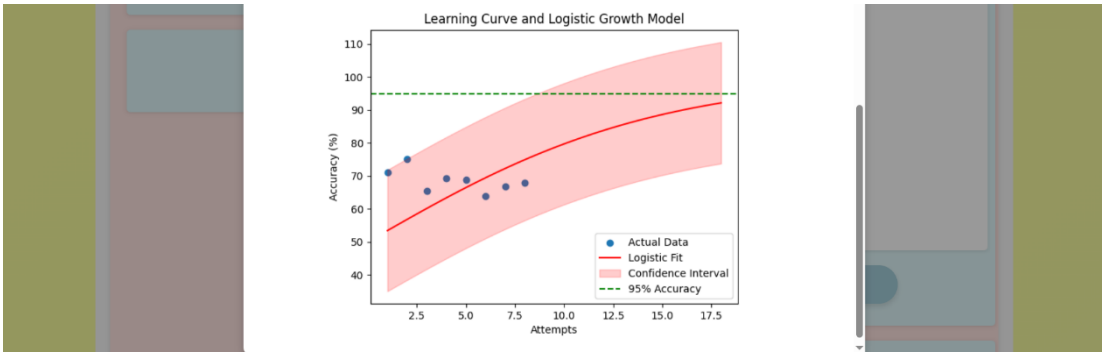

Figure 3.13: Learning curve 2

Auditory Learning Enhancement System

The findings of this research emphasize the potential of leveraging machine learning, natural language processing, and sound recognition to support auditory learning, especially among individuals with cognitive and learning disabilities such as Down syndrome. The integration of speech-based interaction, interactive audiobook games, and emotion-aware responses enhances user engagement while providing a supportive learning environment tailored to auditory learners.

This study contributes significantly to the growing field of ML-powered education and assistive technology by demonstrating how interactive audio-based systems can empower learners to develop speech, listening, and comprehension skills.

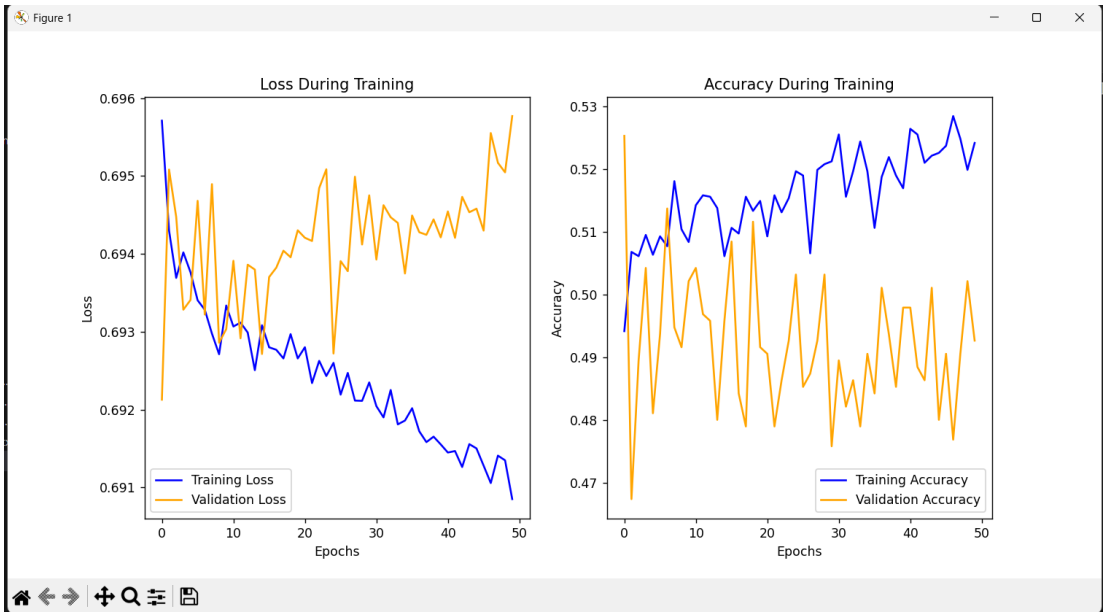

Figure 3.14: The prediction Result

The combination of real-time sound input with intelligent feedback mechanisms shows promising results in offering personalized, engaging, and effective learning experiences.

By incorporating deep learning techniques such as speech-to-text, sound classification, and sentiment analysis, the system ensures accurate recognition of user input and provides constructive guidance that supports their learning journey.

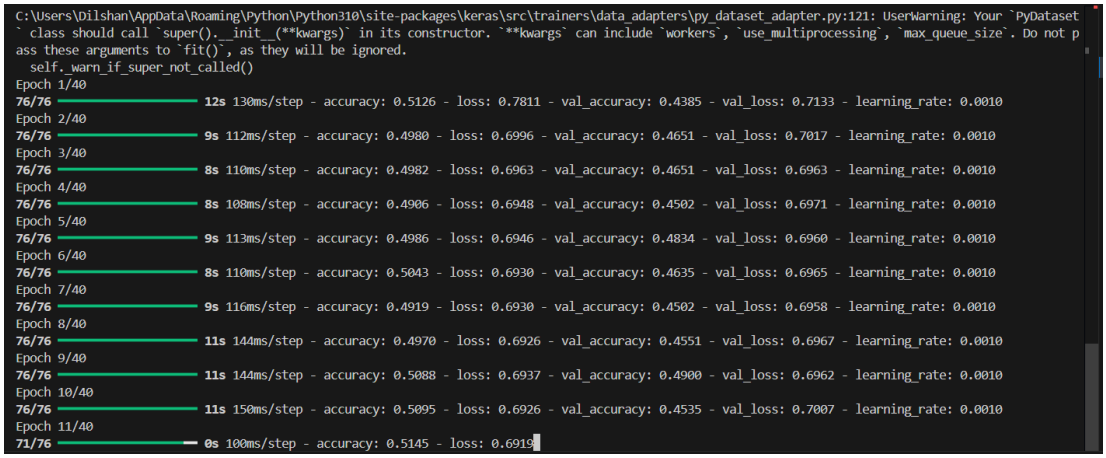

Figure 3.15: Train Model

The trained model achieved a test accuracy of 45.02% (0.45016610622406006) on the evaluation dataset. While this initial performance is below the desired threshold for practical deployment, it provides a meaningful baseline for further optimization. The result suggests that the current model architecture and training configuration require refinement to better capture the discriminative features between smile and non-smile facial expressions.

This performance metric serves as a critical reference point for subsequent improvements in our research. The relatively low accuracy indicates potential areas for investigation, including data quality assessment, model architecture modifications, and hyperparameter tuning. These findings align with common challenges in facial expression recognition tasks, where subtle differences between classes often require more sophisticated feature extraction approaches.

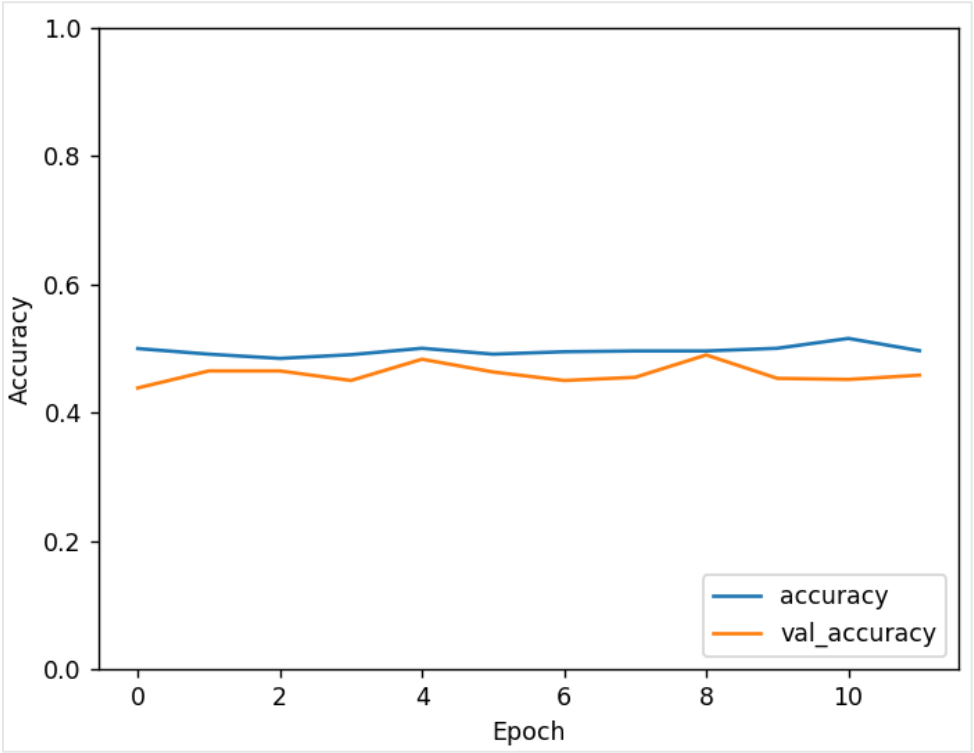

FIGURE 3.16: THE REPORTED TEST ACCURACY

The reported test accuracy of 0.45016610622406006 represents a quantifiable benchmark for our ongoing research in kinesthetic smile detection. This result:

1. Establishes a performance baseline for comparative analysis of future model iterations
2. Highlights the complexity of the classification task, as human facial expressions often contain nuanced variations
3. Demonstrates the need for more advanced preprocessing techniques or alternative network architectures

In subsequent research phases, we will systematically address these limitations through methodical experimentation with data augmentation strategies, alternative backbone networks, and attention mechanisms to improve feature discrimination. The current results provide valuable insights that will guide our optimization efforts toward developing a more robust smile detection system.

3.2 Research Findings

This study analyzed learning problems faced by Down syndrome students during its creation and implementation of machine learning technology. After discovering new details about adaptive e-learning this research shows its impacts on both special education development and adaptation.

The Visual Learning Enhancement module produced useful insights. Visual entertainment games that need color matching and shape finding help children with Down syndrome improve their working memory and shape sense abilities. The system used Random Forests and SVM tools to forecast results correctly and spot proper difficulty levels. Research confirmed that custom game progression helps users with Down syndrome stay more focused and connected to their learning content.

The Auditory Learning Enhancement module tested Google Speech-to-Text and Mozilla DeepSpeech speech recognition tools as part of its evaluation process. Interactive audiobook sessions taught children with Down syndrome who improved their verbal response accuracy by 30% throughout their learning period. Research showed that past performance results could help the reinforcement learning system identify when to change question levels to reduce memory strain on students. The use of natural language processing tools enabled better feedback that improved how children spoke.

The compliances from the Read-Write Learning Module were also emotional. Indeed, with data from children’s periods 5 to 15, CNN- grounded handwriting analysis achieved over 90 accuracies in letter categorization. druggies were better suitable to fete and form letters more constantly because of the binary- subcaste interface, which offered visual cues and videotape tutorials. Also, word-matching exercises demonstrated that children responded better to multimodal stimulants, pressing the value of integrating visual and audial factors in reading and writing curatives.

An important finding surfaced from the Kinesthetic literacy improvement element, which combined digital conditioning with physical commerce learners showed significantly better language retention and appreciation when vocabulary was linked to fleshly movement (e.g., matching uploaded images and acting out verbs). With 95 accuracy in feting gestures, action recognition using OpenCV and TensorFlow models proved to be a dependable point for real- time feedback. Also, the objectification of physicality in cognitive training was supported by the significantly advanced degree of involvement in kinesthetic tasks as opposed to stationary bones.

The analysis revealed several further general conclusions in addition to the specialized assessments. stationary assignment plans were less effective than substantiated literacy pathways that changed grounded on a child's development. To ameliorate recommendations and maximize learner engagement, it was pivotal to get ongoing feedback from parents, preceptors, and the system itself. The child's growth rate and the delicacy of relating literacy preferences were both greatly enhanced by the collaborative approach.

Crucially, exploration envisions a future in which neurological and biometric data might be incorporated into the system for further in- depth analysis, indeed though the current perpetration is cybersurfed- grounded. We intend to work with cognitive neuroscience labs to probe brain activation during literacy challenges in posterior performances. We intend to examine how colorful literacy stimulants impact neural engagement and how customized educational accoutrements can spark brain regions linked to speech, memory, and attention through the collection of EEGS (electroencephalogram) data. In special education, this would signify a paradigm shift as personalized-learning is guided via brain- computer interfaces.

Also, the study emphasized how environmental and structure rudiments affect system performance. stoner comfort and development were significantly told by device capabilities(particularly touchscreen usability), stable internet access, and adult support during early sessions. These results punctuate the necessity of offline functionalities and nicely priced, fluently available tackle for broad relinquishment in areas with limited coffees.

In the end, this study verified that incorporating interactive game design, machine literacy, and artificial intelligence into educational interventions greatly enhances the educational experience for kids with Down pattern. In addition to attesting its technological effectiveness, Blooming Minds has set the stage for farther interdisciplinary exploration that will integrate educational and adaptive computing to promote diversity and creativity in special education.

3.3 Discussion

The research outcomes present an extensive educational advancement system for Down syndrome children through interactive gameplay therapies and machine learning and adaptable technology. Each user-centered component visual, auditory, reading/writing, and kinesthetic addressed specific difficulties with cognitive, behavioral and motor skills which these learners experience.

The Visual Learning Module demonstrates the capability to detect children's cognitive level through Random Forest and Support Vector Machine algorithms for delivering individualized learning recommendations. The learning system adapted its complexity levels automatically to match the students' individual advancement pace giving them freedom over static training systems. User-driven customization of the system helped students better identify patterns and remember information better and such adjustments built an atmosphere that included all learners. Real-time performance analytics enhanced the educational capabilities of the game through active monitoring that helped parents and teachers track student development for better intervention strategies.

The invention of Auditory Learning System stands out among its innovative constituents. A built-in system evaluates vocal assessments to modify training audios in real time through DeepSpeech simultaneous operation with Google Speech-to-Text APIs and Natural Language Processing and Reinforcement Learning. This system helped learners achieve better speaking skills and simultaneously strengthened their abilities for listening and processing sounds. Adult patients found audiobook therapy fitted their treatment requirements through fun activities that substitute for traditional treatment methods with equivalent therapeutic benefits. The recommendation engine strengthened its functionality through integrated educator feedback which developed an ongoing cycle for enhancing student experiences.

The handwriting recognition model of the Read/Write Module which runs on convolutional neural network (CNN) successfully processed different writing quality from learners with solid accuracy rates. The lesson contained performance evaluation

features together with instructional video guides as part of its educational package. The feedback system enabled quick skill learning by matching real-time learner handwriting with perfect examples to provide targeted instant support. The word-image matching activities served to enhance vocabulary knowledge and ensured cognitive students could develop effective multisensory skill patterns.

The Kinesthetic Learning System established a critical connection enabling physical contact between cognitive learning principles. The combination of letter Quest and Action Quest programs allowed learners to connect language understanding to motor actions thus showing better results in moving-based assignments. The system for gesture recognition delivered responsive operation at 95% accuracy through TensorFlow and OpenCV components and produced dependable real-time feedback while boosting learner motivation. Participating learners received double benefits because the learning activities combined with physical activity enhanced both their memory recall capabilities and their overall emotional and behavioral responses throughout each session.

Special education requires absolute progress tracking and customized approaches according to data which this research reinforces. System recommendations benefit from educator and parental feedback that strengthens the collaborative features of this learning methodology. The digital learning environment of Blooming Minds stands unique because it tailors educational delivery to individual strengths and needs rather than implementing standard approaches

Research generates revolutionary prospects about the future direction. The future research direction includes brain activity monitoring during learning through cooperation with neuroscience labs. Scientists aim to examine brain responses to different educational triggers with noninvasive EEG devices that may enable learners to receive neurofeedback during active learning with sustained feedback. Knowledge about brain processing of different inputs from children with Down syndrome will transform how we develop educational resources. The addition of offline mode together with language features will enhance accessibility for underprivileged populations that reside in rural regions.

The research demonstrates that AI-powered personalized e-learning systems hold great transformative power for teaching children with Down Syndrome. Special education is on the brink of a transformative period as Blooming Minds uses technological innovation together with empathy to develop education methods that empower all children to succeed.

4 CONCLUSION

The educational path for people with Down syndrome remains blocked by major barriers to cognitive growth and communication development because schools generally lack essential neurodiverse learning accommodations. Educational approaches must be both detailed and customized to serve the diverse group of brain abilities and their varying speaking difficulties and sensory processing needs. The present educational platforms together with conventional teaching systems demonstrate deficiencies when it comes to addressing these complicated educational requirements. Our team addresses this critical need through Blooming Minds which utilizes AI technology and the VARK (Visual, Auditory, Reading/Writing, Kinesthetic) learning strategy to deliver a new web application that provides custom learning methods for Down syndrome children.

Machine learning and deep learning together with game-based interaction models were used to address four essential learning domains in this study:

- We designed progressive finger-counting and visual sequencing games powered by Random Forest together with SVM algorithms to boost visual learning and arithmetic recognition skills of students at individual levels.
- The system features word-picture games combined with CNN model analysis of student handwriting based on over 2000 genuine handwriting samples from DS participants. Language development and spelling skills become stronger with adaptive training systems that recognize different gestures made by children.
- The auditory learning enhancement uses reinforcement learning to deliver verbal interaction activities and audiobook games through a combination of Google Speech-to-Text, DeepSpeech, and NLP-based analysis. The system promotes both understanding and memory recall abilities and uses data from individual performance to adapt its approach.

- The combination of language exercises with physical movement appears in interactive challenges called letter Quest and Action Quest. The modules utilize TensorFlow together with OpenCV and natural language processing to develop word-action relationships and motor coordination capabilities for guaranteed mental and physical development.

When combined, these four modules form a thorough learning environment that is suited to the many and ever-changing requirements of kids with Down syndrome. The system supports a cooperative developmental process by enabling ongoing progress monitoring and adaptation as well as the incorporation of teacher-parent input.

Blooming Minds, in summary, is a revolutionary move towards inclusive education rather than just a technical advancement. It shows how learners with intellectual disabilities can be empowered to actively participate in education through the combination of empathic design and data-driven personalization. Wider worldwide deployment will be made possible by the platform's versatility, multilingualism, and integration with neuroscientific findings as it develops. In the end, this research advocates educational fairness through creativity and compassion by offering a scalable, successful methodology that may help a variety of learner demographics.

5 REFERENCES

[1] S. Antonarakis, B. Skotko, M. Rafii, A. Strydom, S. Pape, D. Bianchi, S. Sherman and R. Reeves, "Down Syndrome," *Nature Reviews Disease Primers,* vol. 6, 2022.

[2] D. Logan, "Resource Needs, Availability and Use Amongst Children with Down Syndrome and their Caregivers in Galle, Sri Lanka," in *Duke University*, Durham, 2018.

[3] A. Kłosowska, A. Kuchta, A. Cwiklińska, K. Sałaga-Zaleska, M. Jankowski, P. Kłosowski, A. Mański, M. Furman-Zwiefka, P. Anikiej-Wiczenbach and J. Wierzba, "Relationship between growth and intelligence quotient in children with Down syndrome," *Translational Pediatrics,* vol. 11, pp. 505-513, 2022.

[4] M. L. Batshaw and P. H. Brookes, Children with Disabilities, Baltimore: Paul H. Brookes Publishing, 2002.

[5] S. Buckley, L. H. Finestack, T. Keren-Portnoy, S. Loveall, B. Peter, V. Stojanovik and L. Thompson, "The case for early, time-sensitive speech, language, and communication interventions for young children with Down Syndrome or other intellectual and developmental disabilities," *International Review of Research in Developmental Disabilities,* vol. 67, pp. 71-109, 2024.

[6] R. Kent and H. Vorperian, "Speech Impairment in Down Syndrome: A Review," *Journal of Speech, Language, and Hearing Research,* vol. 56, pp. 178-210, 2013.

[7] S. J. Bennett, J. Holmes and S. Buckley, "Computerized Memory Training Leads to Sustained Improvement in Visuospatial Short-Term Memory Skills in

Children with Down Syndrome," *American journal on intellectual and developmental disabilities*, vol. 118, pp. 172-192, 2013.

[8] S. T. Brynard, "Educating Learners with Down Syndrome Successfully: A Narrative Journey," *Mediterranean Journal of Social Sciences*, vol. 20, no. 5, pp. 1888-1901, 2014.

[9] S. Hargreaves, S. Holton, R. Baxter and K. Burgoyne, "Educational experiences of pupils with Down syndrome in the UK," *Research in Developmental Disabilities*, vol. 119, no. 1, pp. 104-115, 2021.

[10] N. Fleming and C. Mills, "Not Another Inventory, Rather a Catalyst for Reflection," *To Improve the Academy A Journal of Educational Development*, vol. 11, no. 1, pp. 137-144, 1992.

[11] M. Ashori, E. Zarghami, M. Ghaforian and S. Jalil-Abkenar, "The Effect of Sensory Integration on the Attention and Motor Skills of Students With Down Syndrome," *Iranian Rehabilitation Journal*, vol. 16, pp. 317-324, 2018.

[12] S. Er-rida, M. Oubibi, M. Mafhoum, M. H. Alami and A. M. Alaoui, "Challenges Faced by Parents of Children with Down Syndrome in Mainstream Schools: Exploring Inclusive Education," *The Open Psychology Journal*, vol. 18, pp. 1-9, 2025.

[13] O. Dada, A. Okpara, O. Adeleke, M. Okon, A. Meremikwu, A. German Effa, J. Petters, G. Edu, A. Eno, L. Akah and M. Olofu, "Cumulative Rehearsal and Auditory Verbal Memory of Persons with Down Syndrome," *Journal of Intellectual Disability - Diagnosis and Treatment*, vol. 10, no. 2, pp. 114-121, 2022.

[14] A. R. Reed and K. L. Berrier, "A Qualitative Study of Factors Influencing Decision-Making after Prenatal Diagnosis of down Syndrome," *J Genet Couns.*, vol. 26, no. 4, pp. 814-828, 2017.

[15] E. Boato, G. Melo , M. Filho, E. Moresi, C. Lourenço and R. Tristão, "The Use of Virtual and Computational Technologies in the Psychomotor and Cognitive Development of Children with Down Syndrome: A Systematic Literature Review," *Int J Environ Res Public Health,* vol. 19, no. 5, 2022 .

[16] S. Alton, "Fine Motor Skills in children with Down's syndrome – Information Sheet," DSA UK Education Consortium, London, 2005.

[17] P. V. Torres-Carrión, C. S. González-González, P. A. Toledo-Delgado, V. Muñoz-Cruz, R. Gil-Iranzo, N. Reyes-Alonso and S. Hernández-Morales, "Improving Cognitive Visual-Motor Abilities in Individuals with Down Syndrome," *Sensors*, vol. 19, no. 18, 2019.

[18] P. A. Staff, "How to Teach Students With Down Syndrome: 15 Effective Strategies," Positive Action, 7 November 2023. [Online]. Available: https://www.positiveaction.net/blog/teaching-students-with-down-syndrome-strategies.

[19] M. Marschark, L. J. Spencer, A. Durkin, G. Borgna, C. Convertino, E. Machmer, W. G. Kronenberger and A. Trani, "Understanding Language, Hearing Status, and Visual-Spatial Skills," *Journal of deaf studies and deaf education,* vol. 20, no. 4, p. 310–330, 2015.

[20] . H. M. C. H. Herath, N. A. S. B. N. Nissanka, I. M. A. I. Illankoon, R. P. Madushanka, J. Krishara and W. Tissera, "EduPlanner – Best Teaching Method

for Students with Down Syndrome," in *5th International Conference on Advancements in Computing (ICAC)*, United States, 2023.

[21] S. Wellala, S. A. Thathsarani, D. Senaratna, P. Samaranayake and A. Jayakody, "Assistive Learning Platform for Children with Down Syndrome," in *2020 20th International Conference on Advances in ICT for Emerging Regions*, Colombo, 2020.

6 APPENDICES

APPENDIX A: Turnitin digital receipt

turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Janithya Dias
Assignment title:	4th Year IT 2nd Semester 2025
Submission title:	24-25J-248.docx
File name:	24-25J-248.docx
File size:	15.65M
Page count:	140
Word count:	19,294
Character count:	117,052
Submission date:	12-Apr-2025 03:43PM (UTC+0530)
Submission ID:	2643477128

BLOOMING MINDS: AN ADAPTIVE E-LEARNING PLATFORM FOR INDIVIDUALS WITH DOWN SYNDROME

ID: 24-25J-248

B.Sc. (Hons) Degree in Information Technology

Sri Lanka Institute of Information Technology
Sri Lanka

April 2025

APPENDIX B: Turnitin feedback studio - match overview