AI-Based Food Recognition & Calorie Analysis System

Object-Oriented Programming Mini Project Report

Under the Guidance of:
Mrs. Puja Gudadhe
Adjunct Assistant Professor

Indian Institute of Information Technology, Nagpur

1. Project Information

Project Title: AI-Based Food Recognition & Calorie Analysis System

Team Members:

- Anant Srivastava - BT24CSH034
- Atharva Gaykar - BT24CSH047
- Aditya Singh - BT24CSH051
- Pratham Dwivedi - BT24CSH063
- Hrushikesh Dhanajay Paithankar - BT24CSH044

2. Description of Functionalities Being Implemented

Our project is an integrated AI-powered food search engine that leverages the power of deep learning and object-oriented programming to create a comprehensive food recognition and nutritional analysis system. The application combines a fine-tuned CLIP (Contrastive Language-Image Pre-training) model with a robust Java-based client interface to deliver an intuitive user experience.

Core Functionalities:

Text-Based Food Query System: Users can input textual descriptions of dishes, such as "spicy paneer curry" or "butter chicken masala," and the system retrieves the closest matching dish from our curated database. The system also handles recipe-based queries like "How to make samosa?" or "what is recipe of idli?" making it versatile for both identification and cooking assistance purposes.

Image-Based Food Recognition: The application accepts food images either through file upload or by providing a file path. Our fine-tuned CLIP model analyzes the image and predicts the matching dish with high accuracy. This feature is particularly useful when users have a picture of food but don't know its name or want to verify what they're looking at.

Calorie and Nutrition Analysis: Once a dish is identified, the Calorific Food Quantifier feature computes detailed nutritional information including total calorie count, macronutrient breakdown, and other essential dietary metrics. This helps health-conscious users make informed decisions about their food choices.

Cross-Language Integration: The system demonstrates seamless communication between Java (frontend) and Python (backend) through HTTP requests and JSON data exchange. The Java client sends queries to the FastAPI server, which processes them using the CLIP model and returns structured responses that the Java application can display in a user-friendly format.

Data Retrieval and Display: After receiving the dish ID from the backend, the Java application fetches comprehensive information from the OOPS_DATA.json file, including ingredients, preparation methods, nutritional facts, and serving suggestions. The DisplayDishes class formats this information for clear presentation to the user.

The system architecture follows a modular client-server design where the Java frontend handles user interactions and display logic, while the Python backend manages the computational heavy lifting of image processing and machine learning inference. This separation of concerns exemplifies good software engineering practices and makes the system scalable and maintainable.

---

3. Brief Description of Classes and Objects Being Implemented

Our project demonstrates strong object-oriented design principles through well-structured classes that encapsulate specific functionalities. Here's a detailed breakdown of the key classes and their responsibilities:

Core Java Classes:

NutritionFacts Class: This class serves as a data structure to encapsulate all nutritional information about a dish. It contains attributes like calories, proteins, carbohydrates, fats, vitamins, and minerals. The class implements getter and setter methods following proper encapsulation principles, ensuring that nutritional data can only be accessed and modified through controlled interfaces. This class is instantiated for each dish to store and retrieve its specific nutritional profile.

Dish Class: The Dish class represents a complete food item in our system. It acts as a comprehensive data model containing attributes such as dish name, dish ID, ingredients list, recipe instructions, cuisine type, and an instance of the NutritionFacts class. This demonstrates composition in OOP, where a Dish "has-a" NutritionFacts object. The class provides methods to access dish details, update information, and present the data in a structured format. Each query result instantiates a Dish object populated with data from our JSON database.

DisplayDishes Class: This utility class handles all presentation logic for displaying dish information to users. It contains static methods that format dish objects into readable output, handle console formatting with appropriate spacing and alignment, and manage the display of images if available. The class demonstrates the single responsibility principle by focusing solely on presentation concerns, separating it from business logic and data management.

CalorieCalculator Class: This class implements the core logic for nutritional analysis and calorie computation. It takes a Dish object as input and performs calculations based on serving size, portion adjustments, and ingredient quantities. The class can compute total caloric value, macronutrient distributions, and provide dietary recommendations. It includes methods for calculating calories from different macronutrients using standard conversion factors (4 cal/g for protein and carbs, 9 cal/g for fats).

Backend Query Classes:

Main Class: The Main class serves as the entry point for the Java application. It orchestrates the entire workflow, managing user input, determining query type (text or image), calling appropriate API handlers, and coordinating between different modules. The class implements a menu-driven interface that guides users through various functionalities.

ClipImageTestQuery Class: This class handles all image-based queries. It manages file I/O operations for reading images, converts image data to appropriate formats, constructs HTTP POST requests to the FastAPI server with image data, and processes the returned dish ID. The class demonstrates how Java can interact with external services through RESTful APIs.

CliptestQuery Class: Similar to ClipImageTestQuery but specialized for text-based queries. It accepts user text input, formats it as JSON payload, communicates with the Python backend through HTTP requests, and returns the matching dish ID to the main application flow.

Calorie Class: This class acts as a bridge between the backend API and the CalorieCalculator. It retrieves dish information using the ID obtained from queries, instantiates appropriate objects, and triggers nutritional analysis. It demonstrates how different classes collaborate to achieve complex functionalities.

Object Interactions:

When a user queries the system, the Main class creates instances of query handler objects (ClipImageTestQuery or CliptestQuery). These objects communicate with the backend and return a dish ID. Using this ID, the system creates a Dish object populated with data from OOPS_DATA.json. The Dish object contains a NutritionFacts object that holds all nutritional data. Finally, the DisplayDishes class receives these objects and presents the information in a formatted manner. If calorie analysis is requested, the CalorieCalculator class processes the Dish object and generates detailed nutritional reports.

This architecture demonstrates key OOP principles including encapsulation (data hiding within classes), inheritance (where applicable for extending functionalities), composition (Dish contains NutritionFacts), and polymorphism (through method overloading in display functions). The modular design ensures that each class has a clear, focused responsibility, making the codebase maintainable and extensible.

---

4. Responsibilities of Each Team Member

Anant Srivastava (BT24CSH034)

Role: Integration & Input Handling

Anant served as the integration specialist, focusing on making all the different components work together cohesively. He implemented the interconnections between Java modules, ensuring that Main.java could properly invoke methods from ClipImageTestQuery, CliptestQuery, Calorie, and DisplayDishes classes in the correct sequence. He developed the input handling logic that routes user requests to appropriate handlers based on query type.

Anant worked extensively on the communication layer between the Java client and FastAPI server, implementing HTTP POST request construction, header configuration, and response handling. He handled character encoding issues, request timeouts, and connection error scenarios to make the system robust. His work included developing the menu system that guides users through different functionalities and collecting user inputs with proper validation.

He actively participated in debugging sessions, identifying issues at integration points and optimizing the user interaction flow. His contribution ensured that the system works as a unified application rather than isolated modules.

Atharva Gaykar (BT24CSH047)

Role: Deep Learning & Backend Integration

Atharva took ownership of the entire machine learning pipeline and backend architecture. His primary contribution was fine-tuning the CLIP model specifically for food-related image-text embeddings, which required extensive experimentation with hyperparameters, learning rates, and training data. He developed the FastAPI server (MODEL_API.py) that hosts the model and exposes REST endpoints for inference. Atharva implemented sophisticated retrieval modules including Text_embeddings_oops.py for processing text queries, retrieval_module.py for semantic search functionality, and ClipEmbeddingsDatabaseCreation.py for generating and storing vector embeddings of the food dataset.

On the Java side, he built the API handling logic across multiple classes (ClipImageTestQuery.java, CliptestQuery.java, Calorie.java) ensuring robust communication between the Java client and Python server. He handled error management, timeout configurations, and JSON parsing for API responses. His work ensured seamless cross-language integration, allowing Java objects to leverage the power of deep learning models running on Python infrastructure.

Aditya Singh (BT24CSH051) and Hrushikesh Dhanajay Paithankar (BT24CSH044)

Role: Core Functionality & Output Formatting

Aditya focused on the critical middle layer of the application—data processing and presentation logic. He implemented the JSON parsing mechanisms in Main.java that extract dish information from the OOPS_DATA.json file based on the dish ID returned by the backend. His code handles edge cases like missing data, malformed JSON, and null values gracefully.

He designed the entire output formatting system, creating the DisplayDishes class and related display methods. This involved careful consideration of user experience—determining what information to show, in what order, with what level of detail, and how to make the console output visually appealing and easy to read.

Aditya ensured accurate mapping between retrieved dish IDs and their corresponding details in the database, implementing error checking to verify data integrity.

He also contributed significantly to system testing, creating test cases for different query types, validating output correctness, and refining the display logic based on feedback. His attention to detail in formatting greatly enhanced the application's usability.

Pratham Dwivedi (BT24CSH063)

Role: Feature Development & Data Preparation

Pratham spearheaded the development of the Calorific Food Quantifier feature, implementing the CalorieCalculator.java class that computes and displays comprehensive nutritional metrics. He studied the Mifflin-St Jeor formula that estimates Basal Metabolic Rate (BMR to calculate macronutrient distributions, daily value percentages, and dietary recommendations based on standard nutritional guidelines). His work involved understanding nutritional science principles and translating them into programmatic logic through the CalorieCalculator class, which handles complex calculations for caloric values and nutritional breakdowns.

Beyond coding, Pratham invested significant effort in data collection and curation. He researched various Indian and international dishes, compiled their nutritional information from reliable sources, and organized this data into a structured format. He created and maintained the OOPS_DATA.json file, ensuring consistency in data structure and accuracy in nutritional values. This dataset forms the backbone of the entire system's knowledge base. Additionally, Pratham contributed to documentation, helping prepare the final project report and user guidelines.

---

5. Comparison with Existing Similar Applications

While we didn't conduct extensive competitive analysis, we are aware that similar applications exist in the market, primarily focused on calorie tracking and nutrition management. Let's examine how our project compares conceptually:

MyFitnessPal is perhaps the most well-known calorie tracking application. It has an extensive food database and barcode scanning capabilities. However, it primarily relies on user input and pre-existing database entries. Our project differentiates itself by using AI-powered image recognition, allowing users to simply take a photo rather than manually searching and entering food items.

Calorie Mama and **Snap Calorie** are AI-based food recognition apps that use image recognition for calorie estimation. While these apps share our core functionality of image-based food identification, our project goes beyond by also supporting text-based queries and recipe searches. Our system is more educational in nature, providing detailed ingredient lists and preparation methods alongside nutritional information.

Edamam and **Nutritionix** offer robust APIs for nutritional data but don't provide the integrated user experience we've built. Our project combines the power of modern NLP through CLIP model fine-tuning with a user-friendly interface, creating an end-to-end solution rather than just an API service.

What sets our project apart:

- **Multimodal Querying:** Unlike most apps that focus on either text or image input, we support both seamlessly through our fine-tuned CLIP model
- **Recipe Integration:** We don't just identify food; we provide recipes and cooking instructions
- **Educational Focus:** Our system is designed as a learning tool that teaches users about ingredients and preparation methods

- **Open Architecture:** Our client-server design makes it easy to extend functionality or integrate with other systems
- **Custom Model:** We fine-tuned our own CLIP model on food-specific data, potentially offering better accuracy for our use case compared to generic models

Where we could improve based on market standards:

- Commercial apps have databases with millions of food items; ours is currently limited to our curated dataset
- They offer mobile applications with camera integration; we currently support only desktop usage
- Many apps include social features, meal planning, and progress tracking that we don't currently have
- Commercial solutions have refined UX/UI with visual dashboards; ours is currently command-line based

However, it's important to note that this is an educational project demonstrating OOP principles and AI integration, not a production-ready commercial application. For a mini project, our implementation successfully showcases modern software engineering practices and emerging AI technologies.

---

6. Possible Improvements If Given More Time

1. Mobile Application Development: Converting our command-line interface into a native mobile application (Android/iOS) would significantly improve user experience. This would include integrating the device camera for real-time food recognition, creating intuitive touch-based interfaces, and implementing offline functionality for basic queries.

2. Expanded Food Database: Currently, our dataset is limited to a curated selection of dishes. We could expand this to include thousands of food items from various cuisines worldwide, different preparation styles for the same dish, and regional variations in ingredients and recipes. We could also integrate with public nutritional databases like USDA's FoodData Central for comprehensive coverage.

3. Portion Size Recognition: Enhancing our model to not just identify food but also estimate portion sizes would make calorie calculations far more accurate. This could involve depth perception, reference object comparison, or user calibration features.

4. Meal Planning and Tracking: Implementing a personal dashboard where users can log their daily meals, track nutritional intake over time, set dietary goals, and receive personalized recommendations would add significant value. This would require implementing a user authentication system and a database to store user-specific data.

5. Dietary Preference Filters: Adding support for various dietary requirements (vegan, vegetarian, gluten-free, keto, etc.) would make the application more inclusive. The system could suggest alternative ingredients or modifications to recipes based on user preferences.

6. Real-Time Model Updates: Implementing a feedback mechanism where users could correct misidentifications would allow us to continuously improve the model. We could periodically retrain the CLIP model with this new data to enhance accuracy.

7. Multi-Language Support: Expanding beyond English to support regional Indian languages (Hindi, Tamil, Telugu, etc.) and other international languages would make the application accessible to a broader audience.

8. Integration with Fitness Trackers: Connecting our application with popular fitness devices and apps (Fitbit, Google Fit, Apple Health) would provide users with a holistic view of their health by correlating food intake with physical activity.

9. Social Features: Adding the ability to share recipes, compare meals with friends, or participate in nutrition challenges could increase user engagement and create a community around healthy eating.

10. Advanced Analytics and Visualizations: Implementing graphical representations of nutritional intake trends, macronutrient ratios, and comparison with recommended daily values would make the data more interpretable and actionable.

11. Ingredient Substitution Suggestions: The system could suggest healthier alternatives for ingredients, helping users make better dietary choices while maintaining similar taste profiles.

12. Restaurant Menu Integration: Partnering with restaurants to integrate their menus would allow users to make informed decisions when dining out, seeing nutritional information before ordering.

13. Voice Interface: Implementing voice commands using speech recognition would make the application more accessible, especially while cooking or shopping for groceries.

14. Improved Model Architecture: Experimenting with newer multimodal models like GPT-4 Vision or specialized food recognition architectures could potentially improve accuracy further.

15. Cloud Deployment with Load Balancing: Moving from a local server setup to cloud infrastructure with auto-scaling would handle multiple simultaneous users effectively and provide better availability.

These improvements would transform our academic project into a production-grade application with commercial viability. However, each enhancement would require significant additional development time, testing, and potentially a larger team with specialized skills in mobile development, UI/UX design, and database management.

---

7. Time Invested in Project Completion

The project was completed over a period of **5 days**, with each team member investing approximately **1.5-2 hours of focused work per day**. This translates to roughly **7-10 hours of total development time per team member**, resulting in an estimated **30-35 hours of collective effort** across the entire team.

Day-wise Breakdown:

Day 1 (2-3 hours total): Initial planning, architecture design, and task distribution. We discussed the project scope, decided on technologies to use, and set up the development environment. This included installing necessary libraries, configuring Python virtual environments, and establishing the repository structure.

Day 2 (2-3 hours total): Data collection and preparation. Pratham led the effort in compiling the food dataset while Atharva started working on the CLIP model fine-tuning process. The rest of the team began implementing basic Java class structures.

Day 3 (3-4 hours total): Core development phase. Atharva completed the FastAPI server setup and model integration. Aditya and Anant worked on the Java client, implementing input handling and API communication logic. This was the most intensive development day with significant code being written.

Day 4 (3-4 hours total): Integration and testing. We connected the Java frontend with the Python backend, debugged communication issues, and tested various query scenarios. This involved fixing JSON parsing errors, handling edge cases, and refining the display output.

Day 5 (3-4 hours total): Final refinements, documentation, and testing. We added the calorie calculation feature, improved error handling, wrote code comments, and prepared the GitHub repository with a comprehensive README. This concluded our work by Pratham writing a detailed report on our project.

Time Distribution by Activity:

- Model fine-tuning and backend development: ~30%
- Java client development: ~35%
- Integration and testing: ~15%
- Data collection and preparation: ~15%
- Documentation and reporting: ~5%

The relatively short development timeline was possible because we worked in parallel on different modules and leveraged existing frameworks and libraries. However, this also meant that some advanced features and optimizations had to be deprioritized to meet the deadline. The focused, intensive approach helped us deliver a functional prototype within the constraints of a mini project, though a longer timeline would have allowed for more thorough testing and feature development. Each team member's individual contribution of 7-10 hours was efficiently utilized through clear task distribution and collaborative problem-solving.

---

8. Lines of Code (LOC)

Our project consists of approximately **1300-1400 lines of code** across both Python and Java implementations. This count includes:

Java Code (~650-700 LOC):

Python Code (~450-500 LOC):

Data and Configuration (~200 LOC):

Analysis:

The codebase is relatively compact, which is appropriate for a mini project. We prioritized clean, readable code with proper function decomposition rather than writing overly verbose implementations. Each class has a focused responsibility, keeping individual files manageable in size.

The Java code is slightly longer than the Python code because it includes more structural elements (getters, setters, constructors) and explicit type declarations as per Java's syntax requirements. The Python backend code is more concise due to the language's expressiveness and the high-level abstractions provided by libraries like FastAPI and LangChain.

The line count could vary slightly depending on how we count certain elements (imports, configuration code, utility functions), but the figures above represent the core functional code that implements our project's features. For a project completed in 7-10 hours per person over 5 days, this represents a good balance between functionality and code quality, demonstrating efficient development practices.

9. Requirements for Deployment

To make this application production-ready and deployable at scale, several critical enhancements and infrastructure changes would be necessary:

Technical Infrastructure:

1. Cloud Hosting: Currently, both the Python backend and Java client run locally. For deployment, we'd need to host the FastAPI server on cloud platforms like Google Cloud Platform (using Compute Engine or Cloud Run), or Azure. The server needs to be accessible via a public IP address with proper domain configuration.

2. Database Management: The current JSON file-based data storage should be replaced with a proper database system. Options include PostgreSQL for relational data (dish information, user data) and vector databases like Pinecone or Weaviate for efficient storage and retrieval of CLIP embeddings. This would significantly improve query performance and scalability.

3. Containerization: Creating Docker containers for both the Python backend and Java client would ensure consistent environments across development, testing, and production. Kubernetes could orchestrate these containers for auto-scaling and high availability.

Security Enhancements:

4. Authentication and Authorization: Implementing a robust user authentication system (OAuth 2.0, JWT tokens) to secure API endpoints and track user-specific data. This includes user registration, login, session management, and role-based access control.

5. Input Validation and Sanitization: Strengthening input validation to prevent SQL injection, XSS attacks, and other security vulnerabilities. Implementing rate limiting to prevent abuse and DDoS attacks.

6. API Key Management: If offering the service through an API, implementing secure API key generation, distribution, and rotation mechanisms.

Application Enhancements:

7. GUI Development: The current CLI interface needs to be replaced with; Web interface using frameworks like React or Vue.js for the frontend; Mobile applications for Android (using Kotlin/Java) and iOS (using Swift)

8.. Error Handling and Logging: Comprehensive error handling throughout the application with meaningful error messages. Implementing centralized logging (using tools like ELK stack or Splunk) for monitoring, debugging, and analytics.

9. Caching Strategy: Implementing Redis or Memcached for caching frequently accessed data, reducing database load and improving response times.

10. Content Delivery Network (CDN): Using CDNs like Cloudflare or AWS CloudFront to serve static assets (images, stylesheets) efficiently across different geographic locations.

10. Repository Link

The complete source code, documentation, and project files are available on GitHub at:

https://github.com/atharva-Gaykar/OOPS_PROJECT

The repository includes:

- All Java source files organized in appropriate package structure
- Python backend code including the FastAPI server and model integration scripts
- OOPS_DATA.json containing the food database
- README.md with setup instructions and usage guidelines
- Documentation files and project diagrams
- Requirements.txt and dependency specifications for easy environment setup

We encourage anyone interested in exploring our implementation to visit the repository, clone the project, and experiment with the code. Contributions and suggestions for improvements are welcome through GitHub issues and pull requests.

Conclusion

This project successfully demonstrates the integration of modern AI technologies with object-oriented programming principles to solve a practical problem in the health and nutrition domain. Through the development of this food recognition and calorie analysis system, we achieved several learning objectives:

We gained hands-on experience with advanced concepts in deep learning, particularly multimodal models like CLIP, and understood how to fine-tune pre-trained models for specific domains. The project strengthened our understanding of OOP principles including encapsulation, inheritance, composition, and abstraction through real-world implementation in Java.

We learned valuable lessons about software architecture, particularly the challenges and best practices of building client-server systems that integrate different programming languages. The experience of designing APIs, handling HTTP communication, and managing data serialization/deserialization has been invaluable.

Working as a team on a project with clear module boundaries taught us about collaborative development, version control with Git, and the importance of good communication and task distribution. Each team member developed expertise in their assigned domain while also understanding how their work fits into the larger system.

The project also highlighted the gap between academic implementations and production-ready systems. While we built a functional prototype, we learned about the numerous additional considerations required for deployment including security, scalability, user experience design, and regulatory compliance.

Looking forward, this project serves as a strong foundation that could be extended in numerous directions—from adding more sophisticated features to deploying it as a public service. The modular architecture we've built makes such extensions feasible without requiring major refactoring.

Overall, this mini project successfully achieved its goal of creating a meaningful application that combines theoretical knowledge with practical implementation, giving us confidence in our ability to tackle more complex software engineering challenges in the future.