GSoC Proposal
AI Teaching Assistant or TA Bot

Bikash Boro

<bikashwork17@gmail.com>

<http://github.com/bikas295>

<*omegaUp username = biku619*>

This proposal outlines the design and implementation plan for an AI Teaching Assistant bot for omegaUp. The bot will answer clarification questions proactively and reactively, and perform code reviews for student assignments. By integrating an open-source Large Language Model (LLM) with our existing system, we aim to tighten the feedback loop, allowing students to iterate and improve their work more rapidly.

⭐⭐⭐ Furthermore, to enhance our project, I have added a **bonus section*,** where I've cited practices and norms followed at the most prominent software product-based startup ecosystems. I believe it would be a great add-on for our project and push omegaUp into an even more comprehensive platform at the global scale.

**the bonus section has been added at EOD(end of document)*

Technical skills

I am currently a pre-final year student at Indian Institute of Information Technology, Pune. I am majoring in ECE with a minor in Computer Science.

I am a budding open-source and have contributed to many repositories. One of them was Olake by Datazip, which was a part of my college midterm project, where I contributed to their RDS documentation and error-mapping. Some of them are GSOC orgs too, like AOSSIE where I honed their FAQ section for their principle site, EduAid.

Apart from that, I am an enthusiastic competitive programmer who loves tactical analysis of football team systems (just a soccerhead!). I am currently Specialist rated in codeforces and am going to participate in the upcoming ICPC Regionals '25. So, I believe I will be first-hand consumer of the AI assistant created in omegaUp, which would help me in my programming journey.

During my previous internship at AlgoUniversity (a Y Combinator-backed startup), I was into the development of “CodeGenie,” an AI Teaching Assistant integrated into their online judge platform, which evaluated millions of submissions from Indian Olympiad and ICPC aspirants. This experience gave me deep exposure to large-scale code evaluation pipelines, LLM trade-offs, and real-world prompt engineering. Notably, the team I worked with went on to win Facebook HackerCup 2025 and gave a talk at NeurIPS 2024.

Link to omegaUp merged PRs:

PR	Solves Issue	Merged	Approved By	Reference
Fix of Cloning Issue for Windows Users #8043	#8023	Yes	@heduenas, @pabo99	https://github.com/omegaup/omegaup/pull/8024
Complete Removal of Problematic Files #8044	#8023	Yes	@heduenas	-
Make orden field mandatory #8056	#8015	Not yet merged	-	-

This might get updated, so here you can find the updated list of merged PRs

Education

- University name : Indian Institute of Information Technology, Pune

  (Maharashtra, India)

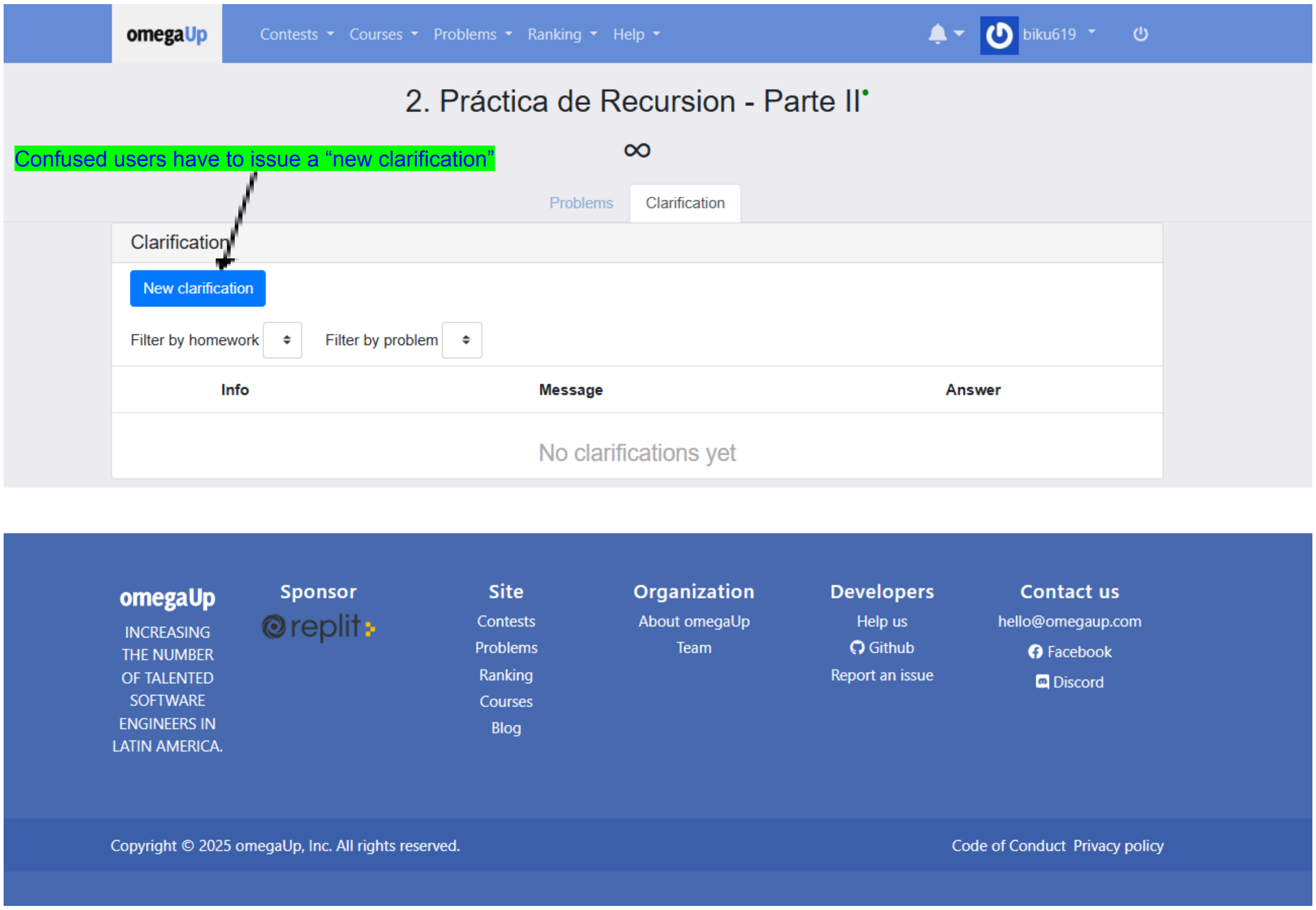
- Major : B.Tech (Bachelor of Technology) in Electronics and Communication Engineering with a minor in Computer Science.
- Starting date: Nov 2022
- Graduation date / Expected graduation date: Jul 2026

Background and Motivation

As a competitive programming enthusiast myself, it brings me immense joy and extra in contributing to a coding platform. It severely aligns with my interests and I don't have to look far away to stay motivated while working on bugs in this project. Furthermore, I agree with omegaUp's ultimate mission to make informatics education available for everyone in Latin America. Honed with skills in development, I believe it is one perfect avenue where I can dedicate my time and skills this summer.

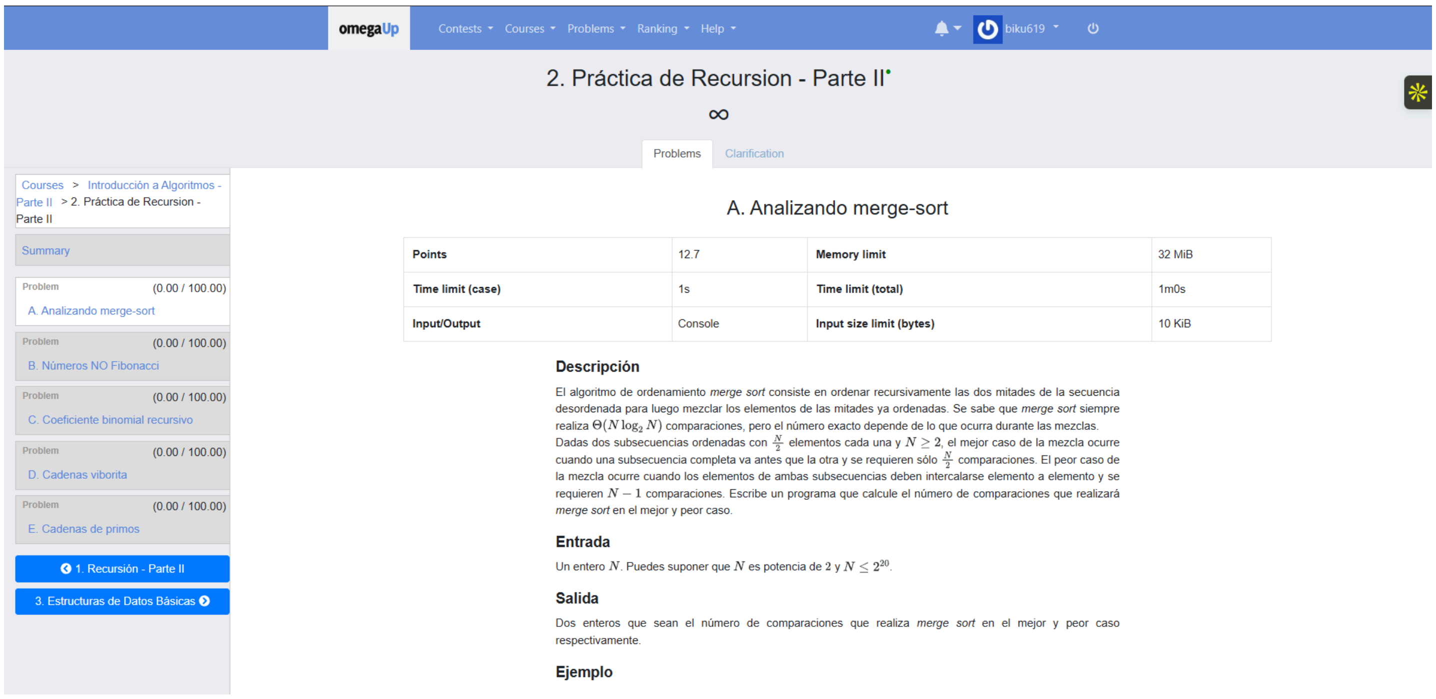
The **current scenario** we are dealing with is that our current (human) Teaching Assistants provide code reviews and answer clarification questions. This **existing role** provides challenges such as human capacity limits, feedback speed, leading to longer response times, which may slow down student progress.

Currently, the only way for a student to resolve their doubt while getting stuck on a course problem is to ask a TA through a clarification forum. But, our human TAs are tentative to reply after some time, which would propose a gap in the students' learnings. Further leaving a doubt unanswered, might leave a student underwhelmed to leave the course. One alternative people have on general problems, available on the site is the solution, with a daily limit of 5*. Such so wouldn't work for course problems.



*as of 26/03/2025

General view of a course problem:


omegaUp
Contests
Courses
Problems
Ranking
Help
biku619
2. Práctica de Recursion - Parte II
∞
Problems
Clarification
Courses > Introducción a Algoritmos - Parte II > 2. Práctica de Recursion - Parte II
Summary
Problem (0.00 / 100.00) A. Analizando merge-sort
Problem (0.00 / 100.00) B. Números NO Fibonacci
Problem (0.00 / 100.00) C. Coeficiente binomial recursivo
Problem (0.00 / 100.00) D. Cadenas viborita
Problem (0.00 / 100.00) E. Cadenas de primos
1. Recursión - Parte II
3. Estructuras de Datos Básicas

A. Analizando merge-sort

Points	12.7	**Memory limit**	32 MiB
Time limit (case)	1s	**Time limit (total)**	1m0s
Input/Output	Console	**Input size limit (bytes)**	10 KiB

Descripción

El algoritmo de ordenamiento *merge sort* consiste en ordenar recursivamente las dos mitades de la secuencia desordenada para luego mezclar los elementos de las mitades ya ordenadas. Se sabe que *merge sort* siempre realiza $\Theta(N \log_2 N)$ comparaciones, pero el número exacto depende de lo que ocurra durante las mezclas. Dadas dos subsecuencias ordenadas con $\frac{N}{2}$ elementos cada una y $N \geq 2$, el mejor caso de la mezcla ocurre cuando una subsecuencia completa va antes que la otra y se requieren sólo $\frac{N}{2}$ comparaciones. El peor caso de la mezcla ocurre cuando los elementos de ambas subsecuencias deben intercalarse elemento a elemento y se requieren $N - 1$ comparaciones. Escribe un programa que calcule el número de comparaciones que realizará *merge sort* en el mejor y peor caso.

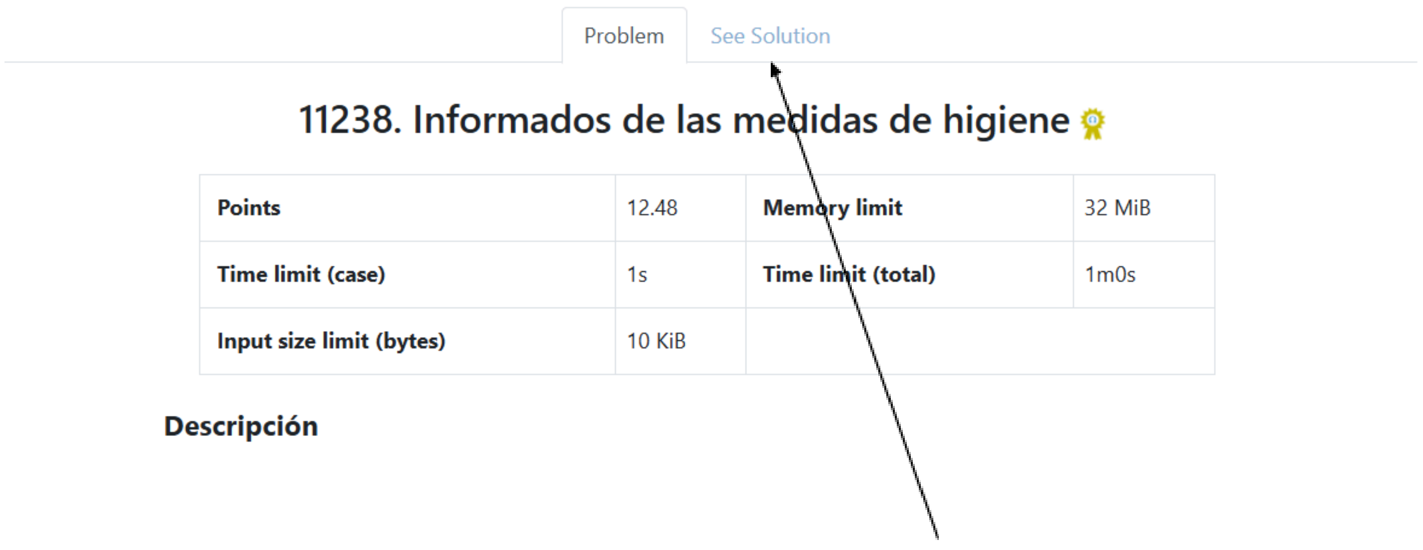
Entrada

Un entero N. Puedes suponer que N es potencia de 2 y $N \leq 2^{20}$.

Salida

Dos enteros que sean el número de comparaciones que realiza *merge sort* en el mejor y peor caso respectivamente.

Ejemplo



11238. Informados de las medidas de higiene

Points	12.48	**Memory limit**	32 MiB
Time limit (case)	1s	**Time limit (total)**	1m0s
Input size limit (bytes)	10 KiB		

Descripción

Seeing the solution is the only way out for confused users for normal problems

My Proposed enhancement would be:

The enhancement involves integrating an AI Teaching Assistant to augment the work of human TAs, not replace them. The AI Assistant will handle common clarification questions and basic code review requests, both proactively and on demand. When it is unable to fully resolve a student's query or identify deeper issues, the system will escalate the matter to a human TA via the existing clarification mechanism—keeping them in the loop and reducing their routine load. This hybrid model ensures scalable support while maintaining high-quality, human-in-the-loop feedback for edge cases and complex doubts.

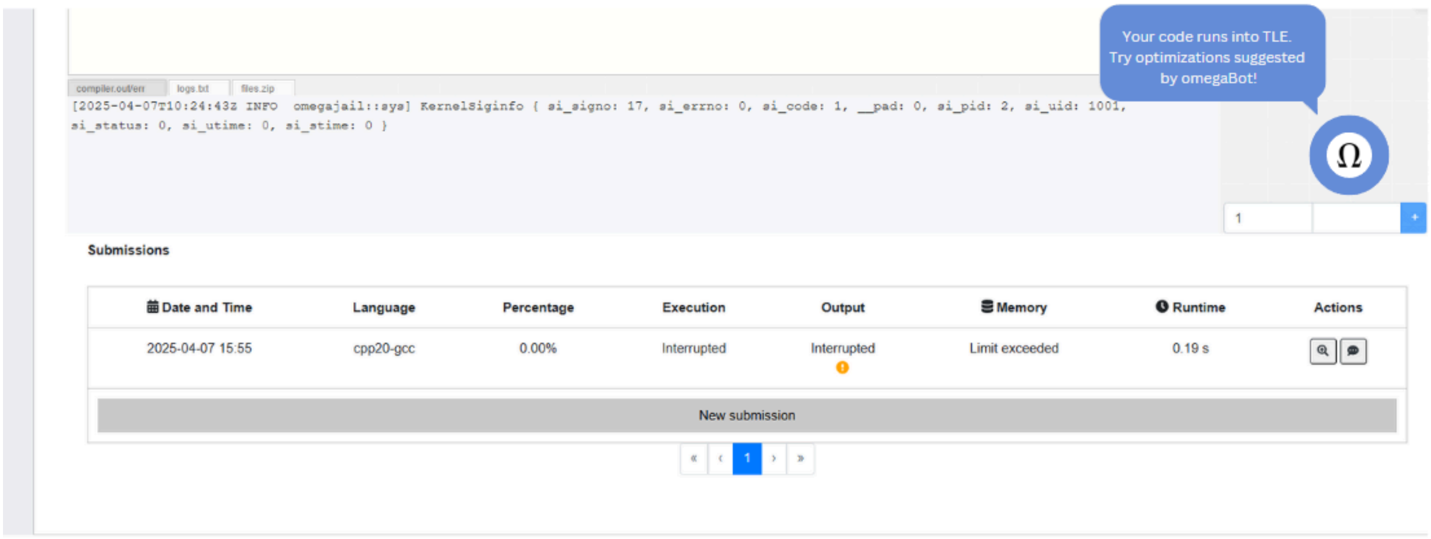
***However, my proposal is directed towards providing this AI Assistant bot for all questions in omegaUp, whether it is related to courses or not. This is ofcourse, a suggestion guided by my principles and virtues of what I believe in should be open to all. This would further enhance omegaUp's consumerism, as it would appeal to more students in the long run. However, I am ready to still work on integrating the assistant for our coursework related tasks, which would follow a very similar process of development.

Detailed project description

Overview

The AI Teaching Assistant aims to enhance the omegaUp platform by providing automated, real-time assistance to students. Leveraging advanced Natural Language Processing (NLP) models, the assistant will offer:

A) Proactive Code Reviews: Analyzing student submissions to provide immediate feedback on code quality, efficiency, and adherence to best practices.



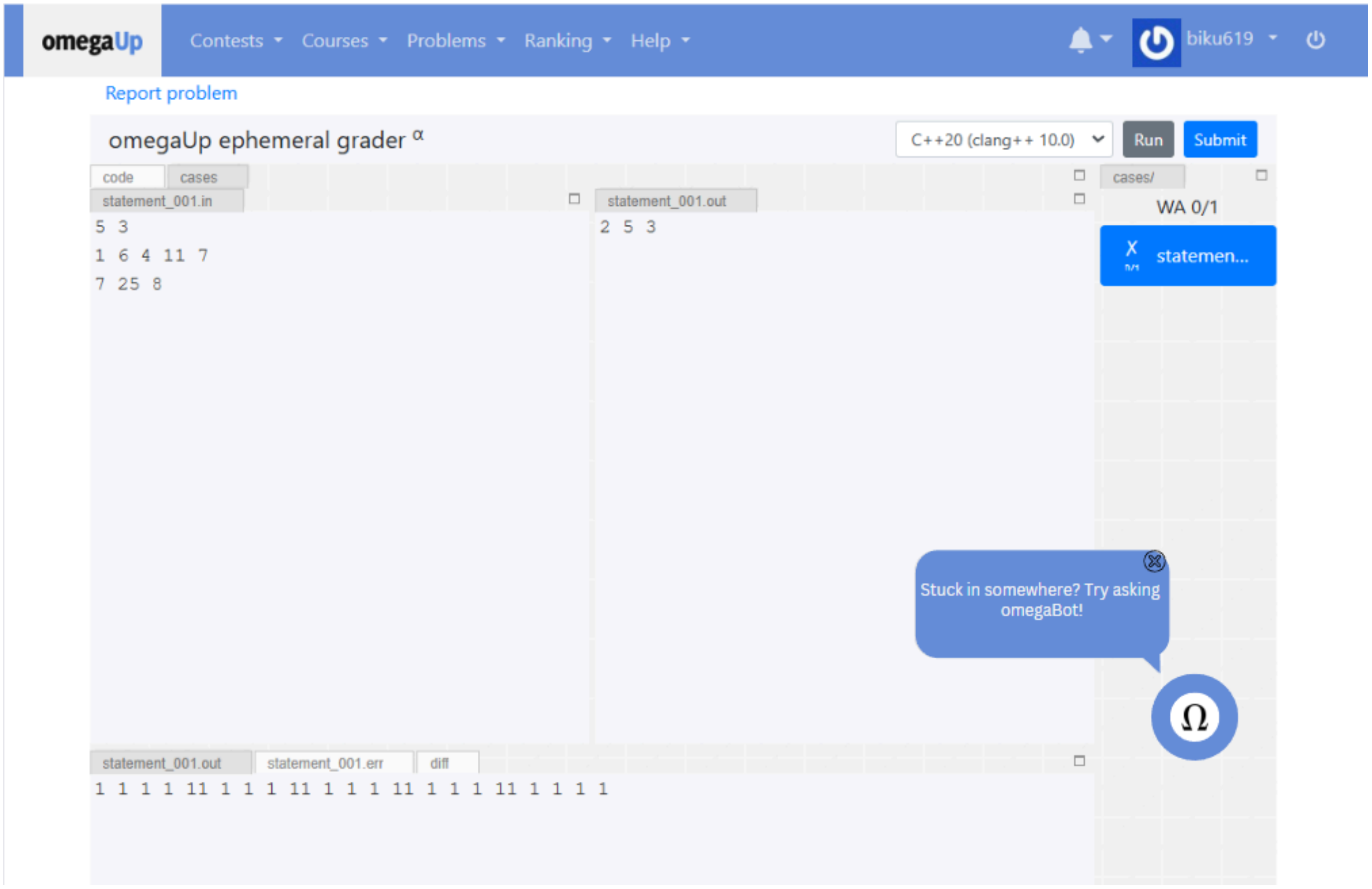
```
[2025-04-07T10:24:43Z INFO  omegajail::sys] KernelSiginfo { si_signo: 17, si_errno: 0, si_code: 1, __pad: 0, si_pid: 2, si_uid: 1001,
si_status: 0, si_utime: 0, si_stime: 0 }
```

Submissions

Date and Time	Language	Percentage	Execution	Output	Memory	Runtime	Actions
2025-04-07 15:55	cpp20-gcc	0.00%	Interrupted	Interrupted	Limit exceeded	0.19 s	

. Our AI assistant immediately prompts up a message upon the student's wrong submission.

B) Responsive Clarification Answers: Addressing student queries related to assignments, problem statements, and coding concepts promptly.



```
5 3
1 6 4 11 7
7 25 8
```

```
2 5 3
```

```
1 1 1 1 11 1 1 1 11 1 1 1 11 1 1 1 11 1 1 1 1
```

Students can use our AI Assistant anytime in . responsive mode by submitting their queries

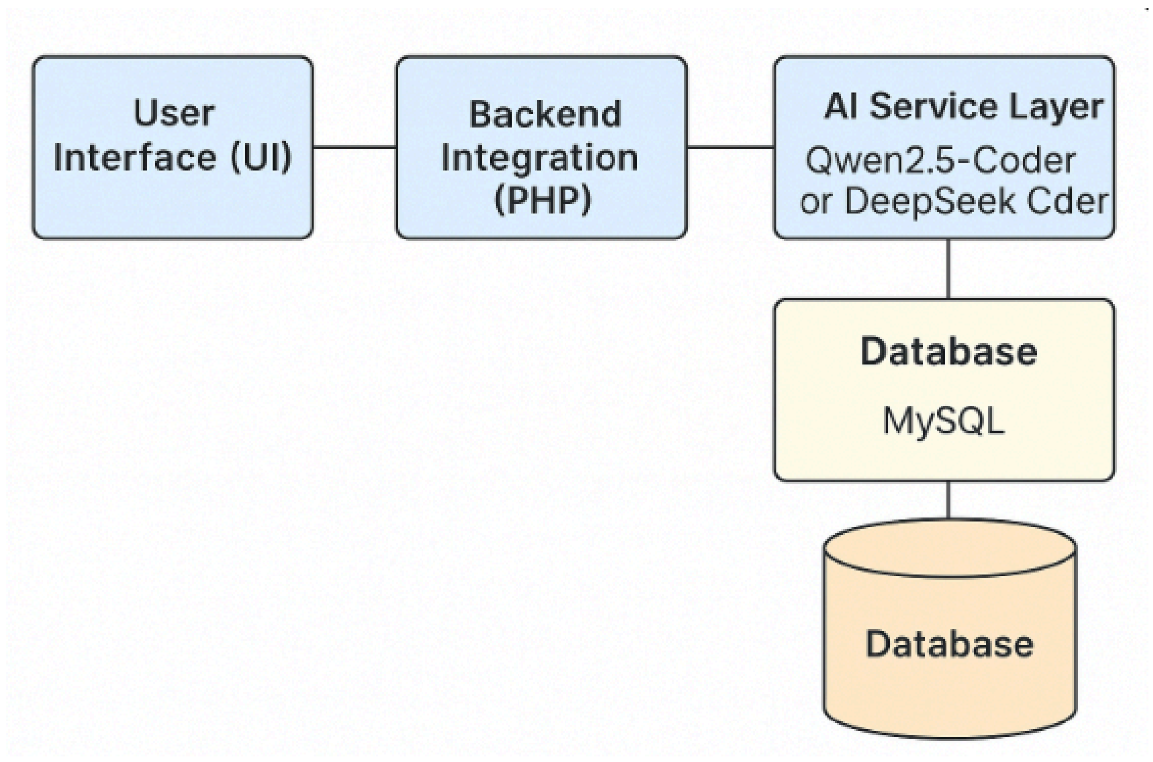
. .

This integration seeks to augment the capabilities of human Teaching Assistants, ensuring scalable and consistent support for the growing user base of omegaUp.

System Architecture

1. **Frontend :** Interactive UI for students with inline AI feedback, chat modes, and real-time code review triggers.
2. **Backend (PHP Layer) :** Routes frontend requests to AI services and handles WebSocket updates for real-time interaction.
3. **AI Service Layer (Python) :** Processes prompts, evaluates models, and manages optimization and test suite executions.
4. **LLM Model Orchestration :** Handles model routing with Qwen2.5-Coder as primary model
5. **Database :** Stores submissions, AI feedback, evaluations, and system metadata using omegaUp's existing MySQL setup.

6. **Integration Strategy :** All AI logic exposed via API for easy embedding across omegaUp interfaces and tools.



Model Selection

My primary recommendation would be Qwen2.5-Coder. This model has shown excellent performance on diverse coding benchmarks (see evaluation results at reference). It achieves 85.4% pass@1 on HumanEval+ and scores 78.9% on MBPP+ with 32B parameters. Also, its 128K token context handles complex codebases.*

In addition, as a fallback and to further enrich our ecosystem, we will support one open source LLM alternative from the same leaderboard. Specifically, for the *Humaneval*+ benchmark, we can integrate OpenAI's model, and for *MBPP*+ we will adopt Google's open source option (if available). Moreover, we plan to incorporate **DeepSeek Coder** as another alternative—leveraging insights and performance limits as outlined in reference. Another open source alternative we can discuss is StarCoder-15B, since it supports superior Python handling (52.7% MBPP score reference) and 8k context. Also it has built-in tech assistant capabilities via prompt engineering.

Deployment Architecture

The updated TA (Teaching Assistant) architecture begins with the Student Interface, where students interact with the system. These interactions trigger the AI TA Engine, which supports two types of invocation: on-demand through a *Code Review Request*, and proactive via automatic *New Submission Scan* after every code submission.

Both these triggers are processed and sent to the Qwen API Analysis module, which serves as the core LLM analysis engine. The output of this analysis is routed through the Feedback Generation component, where structured and insightful feedback is prepared based on the model's analysis.

The generated feedback is then stored in the omegaUp Database, ensuring persistence and traceability. From there, the feedback flows into the TA Dashboard, where mentors and TAs can view, verify, or take further action on the AI-generated content.

Finally, the system includes a Human Oversight layer, allowing educators or senior mentors to audit and refine AI outputs when needed. This ensures a balance between automation and pedagogical quality, enabling a safe and effective assistant experience for learners while maintaining transparency and educator control.

Key User Interface Components

1. Code Submission Page Integration

- Inline code annotations visible in diff view
- "Explain This Suggestion" hover panels

2. Q&A System Enhancement

```
// frontend/server/src/Controllers/ClarificationController.php
public function handleClarification($question) {
    $aiResponse = $this->aiClient->askQuestion($question);
    return $this->mergeResponses($aiResponse, $humanResponses);
}
```

- AI answers marked with β symbol (beta status)
- Confidence score display for answers
- "Still Confused?" escalation path to human TAs

Core Workflows

Proactive Code Review Engine

```
class ProactiveReviewer:
    def analyze_submission(self, submission):
        if self.needs_review(submission):
            review = QwenAPI.generate_review(
                code=submission.code,
                context=submission.problem.rubric
            )
            self.store_feedback(submission, review)

    def needs_review(self, submission):
        return (submission.runtime > threshold
                or submission.language in monitored_langs)
```

Triggers:

- Third-time submission errors
- Performance outliers (runtime/memory)
- Pattern matches from error database

Clarification Handling System

AI Response Pipeline:

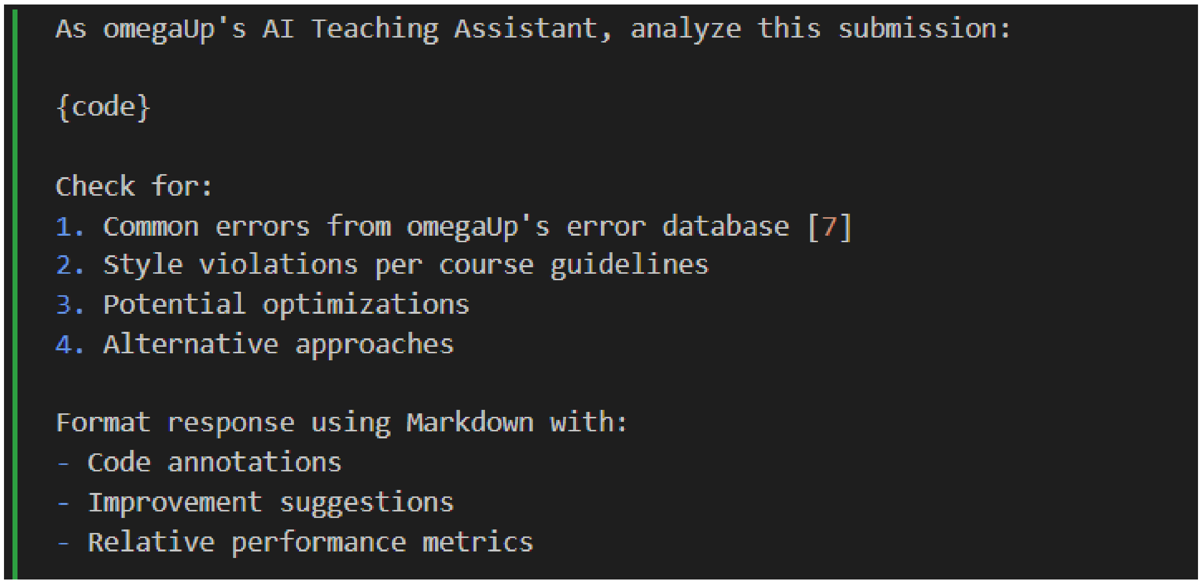
1. Parse question context (problem ID, course section)
2. Retrieve similar resolved clarifications
3. Generate Qwen response with:

```
You are omegaUp's AI TA. Answer questions about {problem} using:
- Course materials up to {week 4}
- Official problem statements
- Approved solution approaches

Question: {question_text}
```

4. Attach confidence score and source references

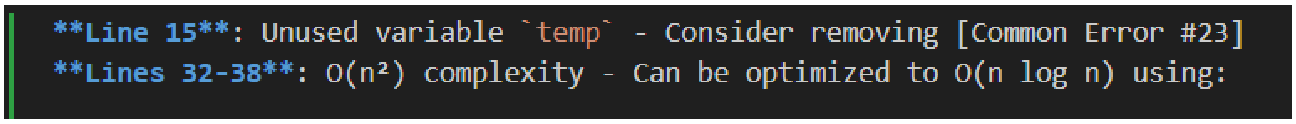
Prompt Engineering Strategy

Code Review Template:



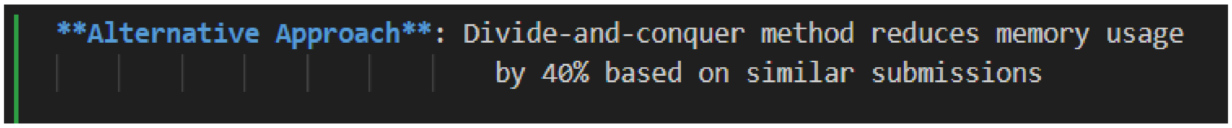
```
As omegaUp's AI Teaching Assistant, analyze this submission:

{code}

Check for:
1. Common errors from omegaUp's error database [7]
2. Style violations per course guidelines
3. Potential optimizations
4. Alternative approaches

Format response using Markdown with:
- Code annotations
- Improvement suggestions
- Relative performance metrics
```

Sample Output Structure:



```
**Line 15**: Unused variable `temp` - Consider removing [Common Error #23]
**Lines 32-38**: O(n²) complexity - Can be optimized to O(n log n) using:
```

sorted_data = heapq.nsmallest(k, data)

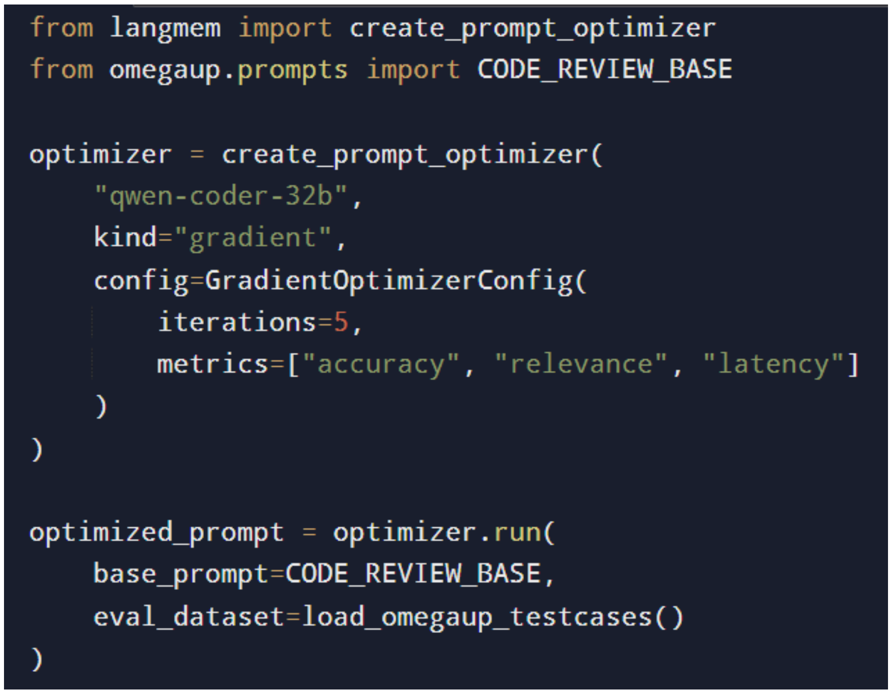


```
**Alternative Approach**: Divide-and-conquer method reduces memory usage
                           by 40% based on similar submissions
```

⭐⭐⭐
Additional suggestions (These are some concepts I've worked hard to design which I think would be beneficial for the project if it could be added on based on suggestions by mentors) :

1. Prompt Strategy Optimizer

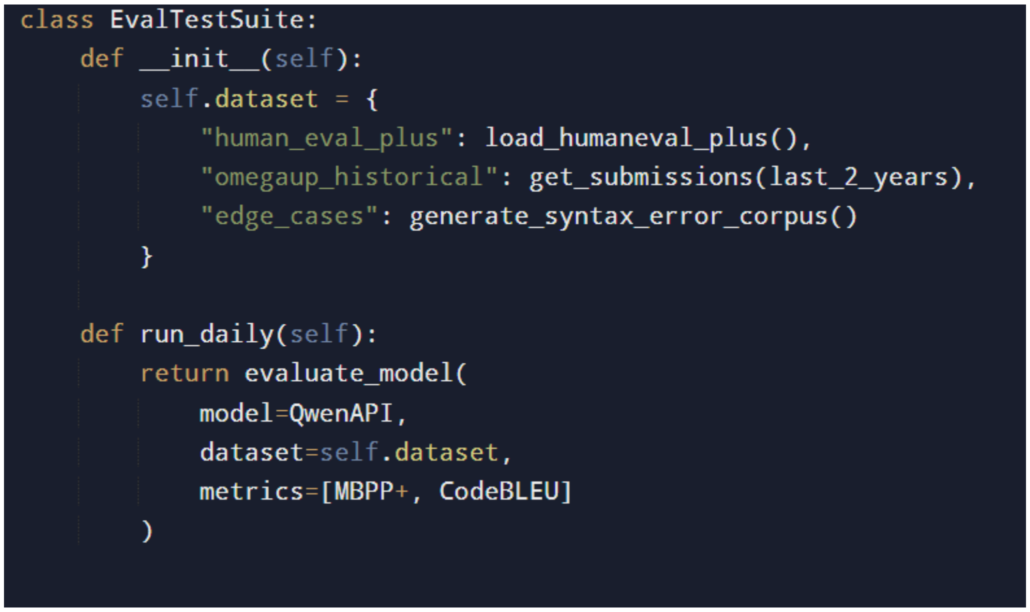
Implementation Approach:


```python
from langmem import create_prompt_optimizer
from omegaup.prompts import CODE_REVIEW_BASE

optimizer = create_prompt_optimizer(
    "qwen-coder-32b",
    kind="gradient",
    config=GradientOptimizerConfig(
        iterations=5,
        metrics=["accuracy", "relevance", "latency"]
    )
)

optimized_prompt = optimizer.run(
    base_prompt=CODE_REVIEW_BASE,
    eval_dataset=load_omegaup_testcases()
)
```

The **Prompt Strategy Optimizer** is designed to automatically fine-tune prompt templates for coding tasks using real evaluation data. By leveraging models like Qwen-Coder-32B and running multiple iterations, it tests prompt variations against metrics such as **accuracy**, **relevance**, and **latency**. This allows for automated A/B testing and ensures the best-performing prompt is selected. It also includes version control for prompt templates, making the system adaptable and easy to maintain. In short, it helps optimize how the assistant communicates—making responses more accurate, relevant, and efficient.

2. Eval-Test Framework

Test Data Composition:

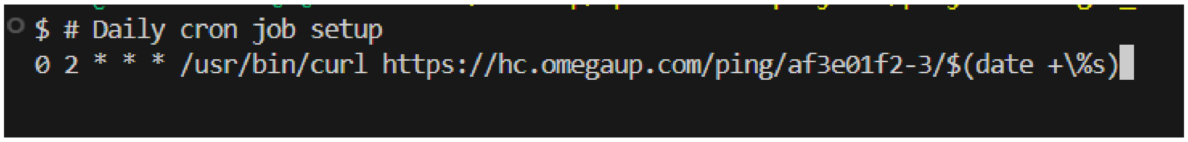


```python
class EvalTestSuite:
    def __init__(self):
        self.dataset = {
            "human_eval_plus": load_humaneval_plus(),
            "omegaup_historical": get_submissions(last_2_years),
            "edge_cases": generate_syntax_error_corpus()
        }

    def run_daily(self):
        return evaluate_model(
            model=QwenAPI,
            dataset=self.dataset,
            metrics=[MBPP+, CodeBLEU]
        )
```

The **Eval-Test Framework** would be a robust system for regularly evaluating model performance using over 500 curated test cases from sources like HumanEval+, omegaUp submissions, and custom edge cases. It runs daily checks across key metrics such as **MBPP+** and **CodeBLEU**, enabling automated detection of bias, regression tracking, and analysis of model explainability. In essence, it ensures consistent, unbiased, and measurable improvements in the assistant's coding performance over time.

3. Health-Check System

Implementation:



```
$ # Daily cron job setup
0 2 * * * /usr/bin/curl https://hc.omegaup.com/ping/af3e01f2-3/$(date +\%s)
```

The **Health-Check System** runs a lightweight daily cron job that pings a health endpoint to confirm system availability and responsiveness. It ensures the AI assistant pipeline is functioning end-to-end. Any failure in this check can trigger alerts via Discord or Slack, enabling rapid response from maintainers. This keeps the platform reliable and operational without manual intervention.

Alert Workflow:

```
graph TD
    A[HealthCheck Failure] --> B{Error Type}
    B -->|Critical| C[Slack #alerts]
    B -->|Warning| D[Discord #monitoring]
    C --> E[PagerDuty Escalation]
    D --> F[Create GitHub Issue]
```

Monitoring Dashboard:

Check	Frequency	Success Criteria
Model API	5-min	200 OK <800ms
Feedback Quality	Hourly	≥90% Accuracy
Cache Health	Daily	95% Hit Rate

Integration Tools:

- Healthchecks.io for cron monitoring
- BuildShip for Slack/Discord automation
- Prometheus + Grafana for metrics visualization

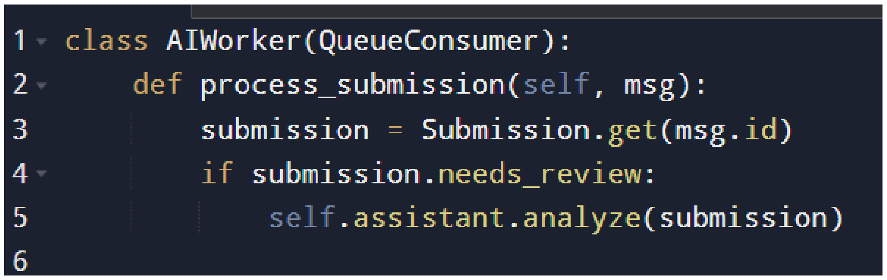
**references for the above are in the appendix as then end of document*

Performance Optimization

1. **Caching Layer**: We can store frequent patterns using omegaUp's existing MySQL infrastructure

```
CREATE TABLE ai_feedback_cache (
    code_hash CHAR(64) PRIMARY KEY,
    feedback TEXT,
    last_used TIMESTAMP
);
```

2. **Proactive Reviews**: Implement background worker using omegaUp's submission queue system

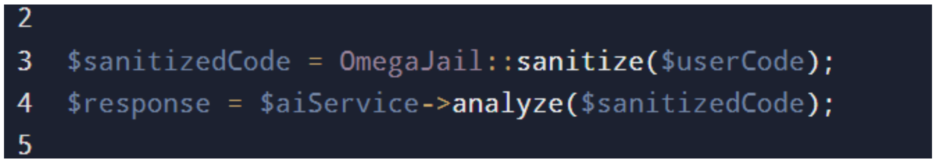


```python
class AIWorker(QueueConsumer):
    def process_submission(self, msg):
        submission = Submission.get(msg.id)
        if submission.needs_review:
            self.assistant.analyze(submission)

```

Security Considerations

1. Code Isolation:

   Route all LLM interactions through omegaJail



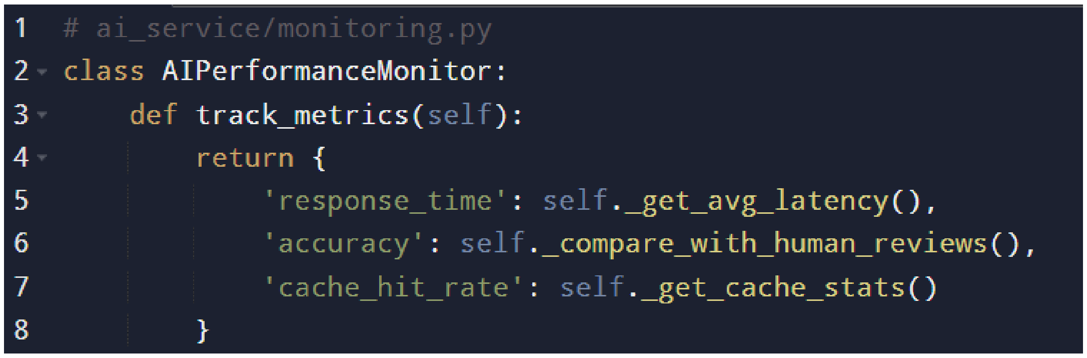
```php

$sanitizedCode = OmegaJail::sanitize($userCode);
$response = $aiService->analyze($sanitizedCode);

```

2. Rate Limiting:

   We can leverage existing rate limiting in `frontend/server/libs/RateLimit.php`

Monitoring Setup

.



```python
# ai_service/monitoring.py
class AIPerformanceMonitor:
    def track_metrics(self):
        return {
            'response_time': self._get_avg_latency(),
            'accuracy': self._compare_with_human_reviews(),
            'cache_hit_rate': self._get_cache_stats()
        }
```

Backend

The backend will undergo several enhancements to accommodate the AI Teaching Assistant:

- API Development**:** Creation of new RESTful API endpoints to handle. Those include **Code Review Requests,** which are endpoints that receive code submissions, forward them to the AI model for analysis, and return feedback. More of these would include **Clarification Queries,** which are Endpoints that accept student questions, process them through the AI model, and deliver responses.

- AI model integration would involve deployment of an open-source LLM, such as Qwen, within the server environment. This would involve Model Hosting, Prompt Engineering and Response Handling.

- Database Schema modifications would involve updating the existing MySQL database to include:

    - **Interaction Logs:** Tables to store records of student interactions with the AI assistant for future analysis and improvement.

    - **Feedback Storage:** Structures to save AI-generated feedback on code submissions, allowing students to revisit past reviews.

- We'll need to implement security measures to ensure data privacy and integrity by implementing Authentication and Authorization. That would involve verifying user identities and controlling access to AI assistant features. Also, we'll need data encryption to protect user sensitive data during transmission and storage.

Frontend

This project will require server-side code changes and selective database updates to support AI-driven interactions on the frontend. The **AI Assistant Interface** will be embedded into omegaUp's submission and problem pages, enabling users to trigger code reviews and receive **inline annotations** rendered using omegaUp's existing markup system. A **progressive hint system** will be implemented with anti-spoon feeding logic to balance learning support. Additionally, a **WebSocket-based pipeline** will be introduced to enable real-time feedback from the assistant.

Design Considerations:

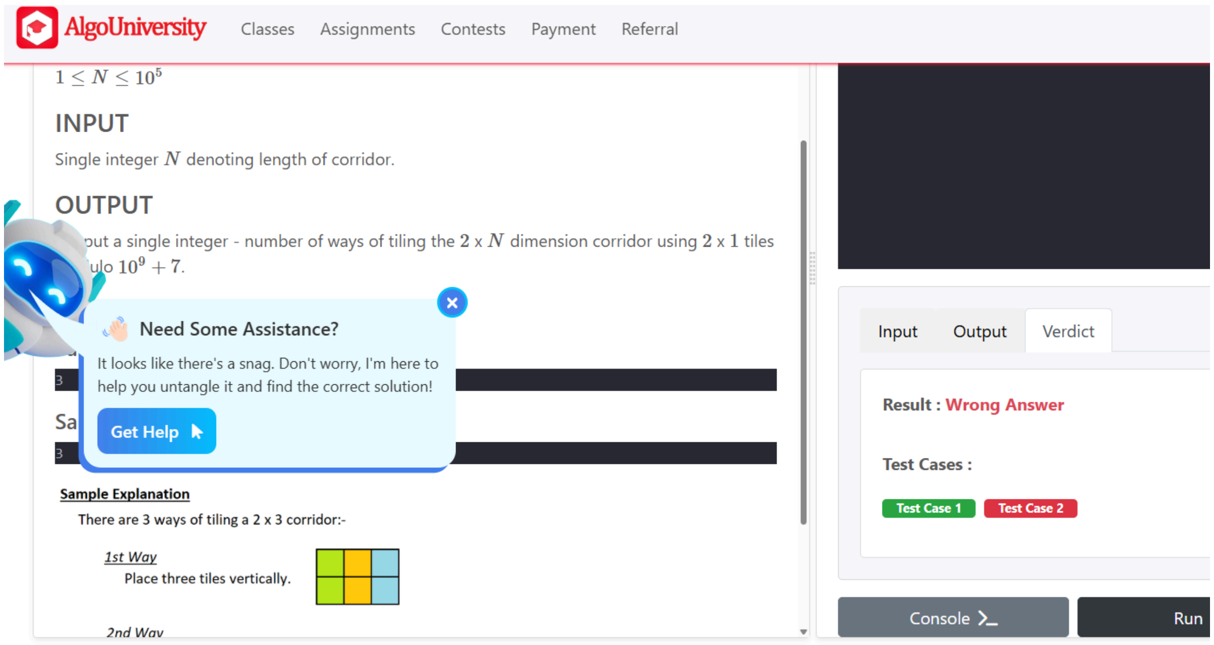
UX is paramount in our design, especially for AI products. Our assistant would only be available for course problems. The assistant will be accessible via:

- Chat Interface

- On-Submission Prompts

- Trigger-Based Notifications

Furthermore, drawing inspiration from projects like Codegenie (the AI chatbot I developed at AlgoUniversity), I suggest we create a distinctive mascot and brand identity. This will help students recognize the assistant as a major feature of the platform. You can refer to examples on **(refer to appendix) for further inspiration.



$1 \leq N \leq 10^5$

INPUT

Single integer N denoting length of corridor.

OUTPUT

put a single integer - number of ways of tiling the 2 x N dimension corridor using 2 x 1 tiles ulo $10^9 + 7$.

Need Some Assistance?

It looks like there's a snag. Don't worry, I'm here to help you untangle it and find the correct solution!

Sample Explanation

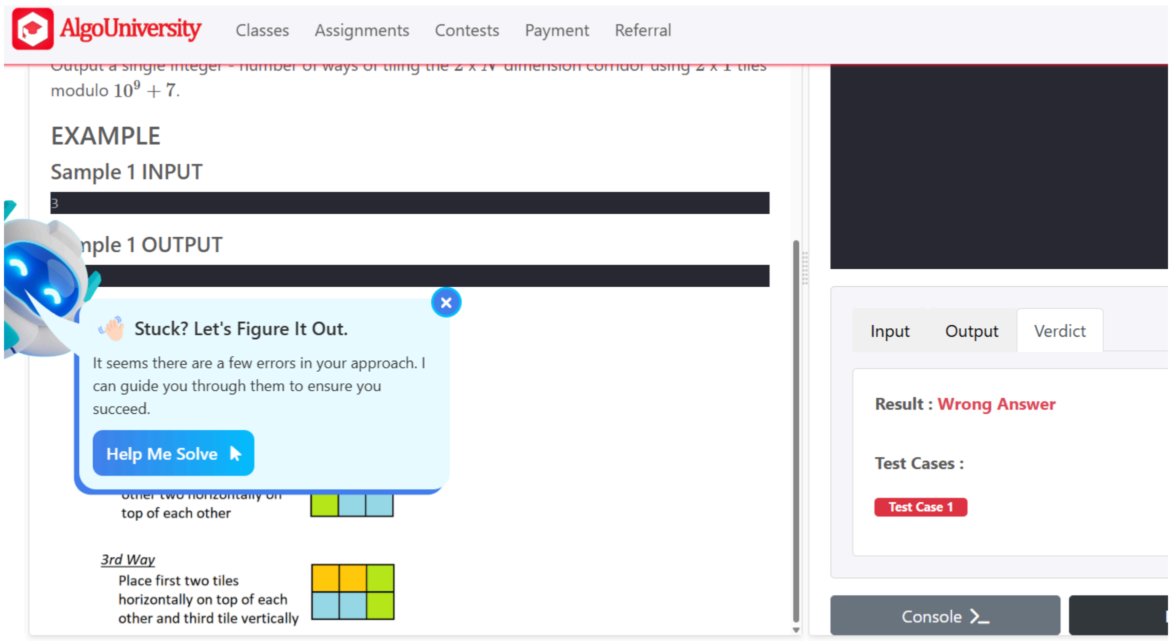
There are 3 ways of tiling a 2 x 3 corridor:-

1st Way
Place three tiles vertically.

2nd Way

Result : Wrong Answer

Test Cases :



Output a single integer - number of ways of tiling the 2 x N dimension corridor using 2 x 1 tiles modulo $10^9 + 7$.

EXAMPLE

Sample 1 INPUT

3

nple 1 OUTPUT

Stuck? Let's Figure It Out.

It seems there are a few errors in your approach. I can guide you through them to ensure you succeed.

other two horizontally on top of each other

3rd Way
Place first two tiles horizontally on top of each other and third tile vertically

Result : Wrong Answer

Test Cases :

Here are the different ways CodeGenie* responds to a student's query: upon wrong submission and upon compliance by the student due to being stuck on a problem for a long time.

User Experience

The AI Teaching Assistant is designed to significantly enhance the user experience on omegaUp by providing prompt feedback on code submissions on **coursework problems**, enabling students to iterate and learn more efficiently. Its 24/7 availability ensures that students can access support at any time, eliminating reliance on human teaching assistants and making learning more flexible. By offering standardized feedback, the assistant maintains consistent quality across all student interactions, reducing discrepancies in guidance.

Additionally, it acts as a learning companion by recommending relevant resources such as tutorials and documentation tailored to each query or submission. Over time, the AI personalizes its interactions, adapting to individual learning styles and pinpointing areas where students may need extra support, fostering a more personalized and effective learning journey.

Alternatives considered

Star-Coder 15B, **DeepSeek Coder**, or any other open source LLM model suggested by mentor. The other domains of the project would remain the same without requiring any alternative.

Security impact

The integration of the AI Teaching Assistant will be designed with strict security measures to prevent misuse and ensure fair competition on omegaUp. The AI will provide guidance based on best practices and general concepts rather than offering direct solutions to problems, reducing the risk of academic dishonesty.

Additionally, to prevent exploitation, my suggestions would be closure of AI Assistant services during contest times (harsh measure) or denial of AI usage on contest problems. However, during coursework on courses available at omegaUp , students can extensively use the AI assistant (obviously with rate limit applied per day).

Deployment plan

Normal deployment + other(s)

A database schema migration will be necessary to store interaction logs, user feedback, and AI-generated responses.
 Furthermore, since the AI assistant will rely on an open-source LLM, it will be deployed using a containerized environment (such as Docker).

*Docker-Based Deployment: We would containerize our LLM application with Docker. I would create a Dockerfile specifying the environment and dependencies for our LLM . Then build and distribute Docker images containing our fine-tuned model.

*Kubernetes for Scalable Deployment: For more complex setups requiring scalability, we can define Kubernetes deployments to manage pods running our LLM . We can then set up services to expose LLM pods to other parts of your application . Then, configure resource requirements carefully (requests and limits) for optimal performance . Also, we can use ConfigMaps for configuration data and Secrets for sensitive information.

Monitoring tools will be integrated to track system performance and detect any issues. After a successful trial period, the assistant will be gradually deployed to all users, ensuring minimal disruption to the existing system.

Schedule

Date	**Milestone**	**Tasks**
May 8	Proposal accepted or rejected	- Community Bonding Period - Discussing implementation details with mentors - Setting up development environment - Learning omegaUp codebase architecture
June 2	Pre-work complete	Start my Project. Long Summer Ahead !
June 9	Milestone #1	- Set up Qwen2.5-Coder API integration - Implement basic prompt templates - Create testing framework for AI responses

June 16	Milestone #2	- Implement code review service - Set up omegaJail sanitization pipeline - Write tests for code review functionality
June 23	Milestone #3	- Build clarification Q&A system - Implement context retrieval from problem database - Write tests for clarification responses
June 30	Milestone #4	- Integrate frontend components (review panel, annotation display) - Implement basic UI for AI Teaching Assistant - Write blog post covering Phase 1 implementation
July 7 - July 14	Phase 1 Evaluation	- Submit midterm report on AI TA core functionality - Begin work on prompt optimization engine, Collect feedback from mentors and early testers
July 19	Milestone #5	- Implement multiple teaching modes (TA, Coach, Mentor) - Create mode-switching UI components - Develop anti-spoon feeding mechanisms
July 26	Milestone #6	- Implement health-check monitoring system - Set up Slack/Discord alerts

		- Add caching layer for common responses
August 2	Milestone #7	- Build prompt strategy optimizer - Implement A/B testing framework - Create mentor dashboard for AI system monitoring
August 9	Milestone #8	- Implement API endpoints for external integrations - Build CodeGenie mascot system and UI - Write second blog post on advanced features
August 19 - August 26	Final Evaluation	- Submit final report on project implementation - Document architecture and integration points - Create user documentation and help pages
September 3 - November 11	Project Extension	- Optimize model performance based on real-world usage - Implement additional features requested by community - Continue monitoring and improving system accuracy

Meeting Notes:

⭐ Had a small conversation with Aritra on 19/03/25 upon the context of the projects and cleared a few of my doubts I had. The conversation mainly revolved around:
1. Hardware constraints: To make informed decisions regarding model selection and fine-tuning

2. Preference on open-source vs closed-source training.

3.Proposal Merge Request: Given the overlap between the AI Teaching Assistant and AI-Generated Problem Editorials projects, we checked the possibility of merging them into a single proposal

⭐ Had a query answered through discord by Juanpa on a database related PR i was working on. He insisted enough time was given on the old PR and I should move on to other issues, which could be reviewed by Hugo Duenas in future.

Bonus Section:

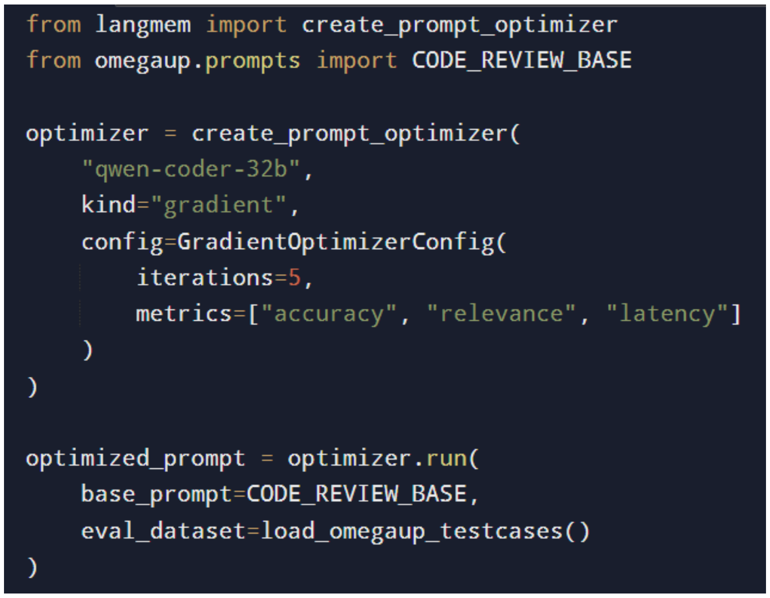
To push the boundaries of accuracy, efficiency, and system robustness, this project includes a bonus suite of enhancements—each aimed at strengthening the assistant's real-world reliability, adaptability, and developer trust. These include a Prompt Strategy Optimizer, an Eval-Test Framework, and a Health-Check System, all designed to ensure continuous learning, measurable improvements, and operational stability.

Also, drawing inspiration from projects like Codegenie (the AI chatbot I developed at AlgoUniversity), I suggest we create a distinctive mascot and brand identity. This will help students recognize the assistant as a major feature of the platform. You can refer to examples on **(refer to appendix) for further inspiration.

Example of mascot in the kattis website

1. Prompt Strategy Optimizer

Implementation Approach:

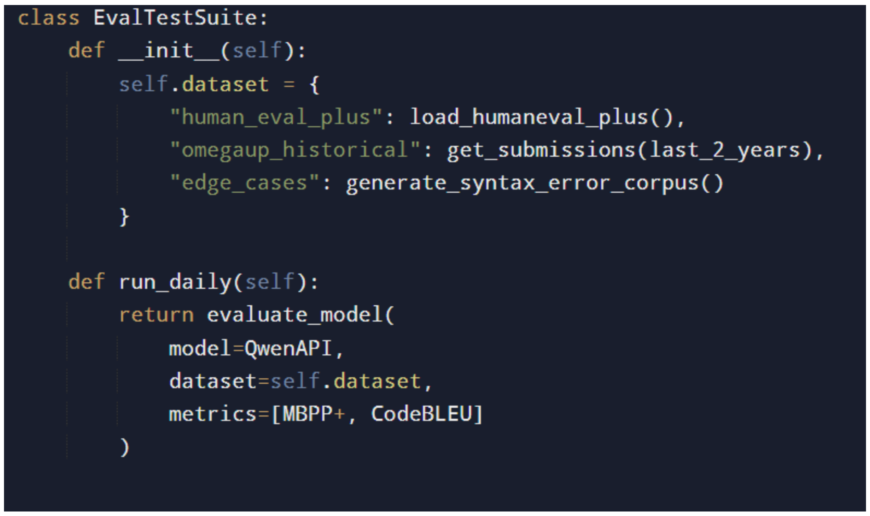


```python
from langmem import create_prompt_optimizer
from omegaup.prompts import CODE_REVIEW_BASE

optimizer = create_prompt_optimizer(
    "qwen-coder-32b",
    kind="gradient",
    config=GradientOptimizerConfig(
        iterations=5,
        metrics=["accuracy", "relevance", "latency"]
    )
)

optimized_prompt = optimizer.run(
    base_prompt=CODE_REVIEW_BASE,
    eval_dataset=load_omegaup_testcases()
)
```

The **Prompt Strategy Optimizer** is designed to automatically fine-tune prompt templates for coding tasks using real evaluation data. By leveraging models like Qwen-Coder-32B and running multiple iterations, it tests prompt variations against metrics such as **accuracy**, **relevance**, and **latency**. This allows for automated A/B testing and ensures the best-performing prompt is selected. It also includes version control for prompt templates, making the system adaptable and easy to maintain. In short, it helps optimize how the assistant communicates—making responses more accurate, relevant, and efficient.

2. Eval-Test Framework

Test Data Composition:

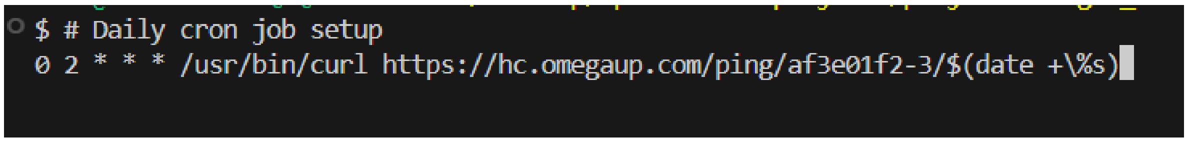


```python
class EvalTestSuite:
    def __init__(self):
        self.dataset = {
            "human_eval_plus": load_humaneval_plus(),
            "omegaup_historical": get_submissions(last_2_years),
            "edge_cases": generate_syntax_error_corpus()
        }

    def run_daily(self):
        return evaluate_model(
            model=QwenAPI,
            dataset=self.dataset,
            metrics=[MBPP+, CodeBLEU]
        )
```

The **Eval-Test Framework** would be a robust system for regularly evaluating model performance using over 500 curated test cases from sources like HumanEval+, omegaUp submissions, and custom edge cases. It runs daily checks across key metrics such as **MBPP+** and **CodeBLEU**, enabling automated detection of bias, regression tracking, and analysis of model explainability. In essence, it ensures consistent, unbiased, and measurable improvements in the assistant's coding performance over time.

3. Health-Check System

Implementation:



```
$ # Daily cron job setup
0 2 * * * /usr/bin/curl https://hc.omegaup.com/ping/af3e01f2-3/$(date +\%s)
```

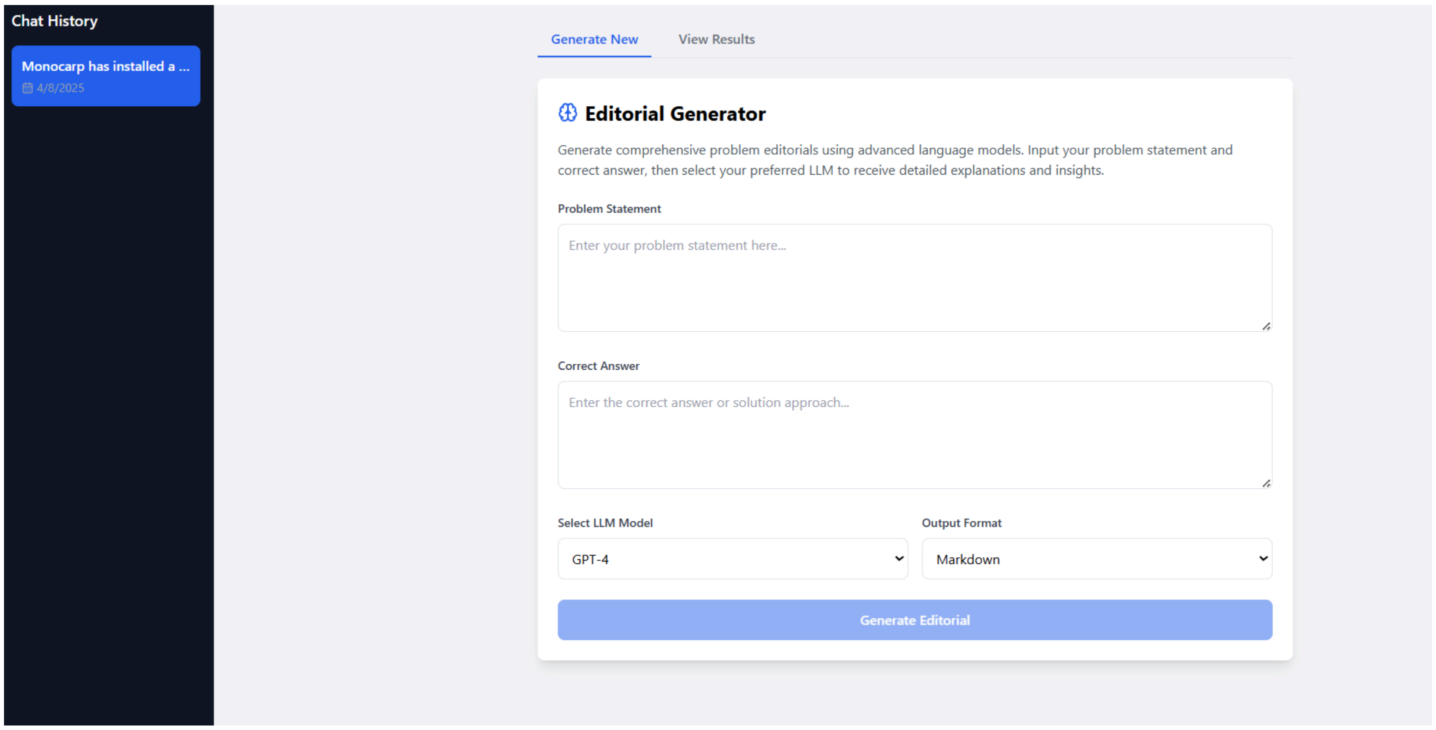
The **Health-Check System** runs a lightweight daily cron job that pings a health endpoint to confirm system availability and responsiveness. It ensures the AI assistant pipeline is functioning end-to-end. Any failure in this check can trigger alerts via Discord or Slack, enabling rapid response from maintainers. This keeps the platform reliable and operational without manual intervention.

⭐⭐⭐Furthermore, I once raised an idea to integrate the top two ideas listed: AI Assistant and Editorial, as both are seemingly co-related and one can be used as input or fodder for the latter.

However, I decided to branch it out into proposals after receiving feedback. But, in the process of that I completed a web-app which **completely** performs the function of editorial generation. I used the app to play with the params, while researching upon the project. This web-app showcases my ability and prowess to deliver the required outcome through efficient and structured coding in an allocated time frame.

Here is the link for access: https://taupe-pasca-cb0b34.netlify.app/ (also have cited it in appendix)

Below is the basic view of my web-app and its core functionalities:



Editorial Generator

Generate comprehensive problem editorials using advanced language models. Input your problem statement and correct answer, then select your preferred LLM to receive detailed explanations and insights.

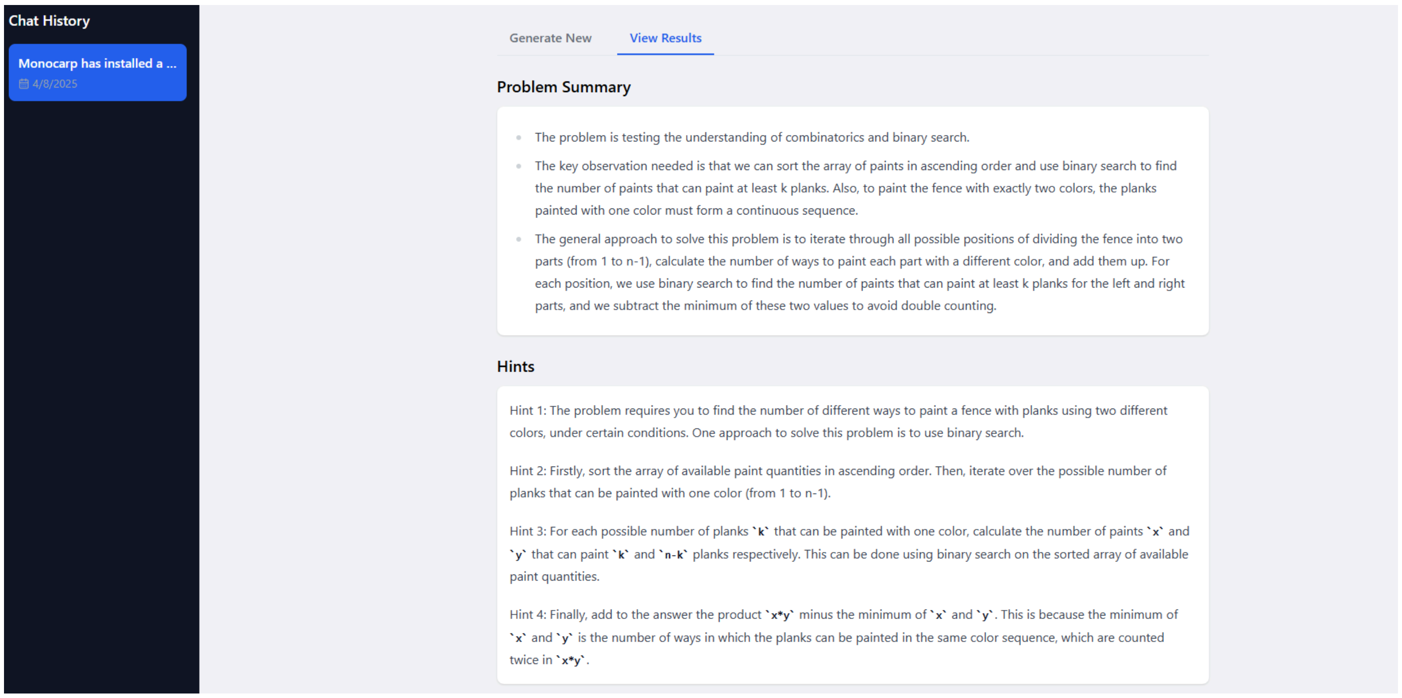
Problem Statement: Enter your problem statement here...

Correct Answer: Enter the correct answer or solution approach...

Select LLM Model: GPT-4

Output Format: Markdown

The **Editorial Generation Interface** is a streamlined web application designed to assist problem setters and reviewers in generating high-quality editorials for competitive programming problems. Built using **Vue.js** for a responsive and modular frontend, it integrates with backend APIs powered by **Python** and **LLMs like Qwen-Coder** to auto-generate detailed problem explanations, time/space complexity analysis, and sample solution breakdowns. Users can input problem metadata, view AI-generated drafts, edit them in-place, and export the final editorial in markdown format. The app supports syntax highlighting, live preview, and version tracking—making it a practical tool for semi-automated editorial curation. The deployment is handled via **Netlify**, ensuring fast, reliable access with minimal setup.



Problem Summary

- The problem is testing the understanding of combinatorics and binary search.
- The key observation needed is that we can sort the array of paints in ascending order and use binary search to find the number of paints that can paint at least k planks. Also, to paint the fence with exactly two colors, the planks painted with one color must form a continuous sequence.
- The general approach to solve this problem is to iterate through all possible positions of dividing the fence into two parts (from 1 to n-1), calculate the number of ways to paint each part with a different color, and add them up. For each position, we use binary search to find the number of paints that can paint at least k planks for the left and right parts, and we subtract the minimum of these two values to avoid double counting.

Hints

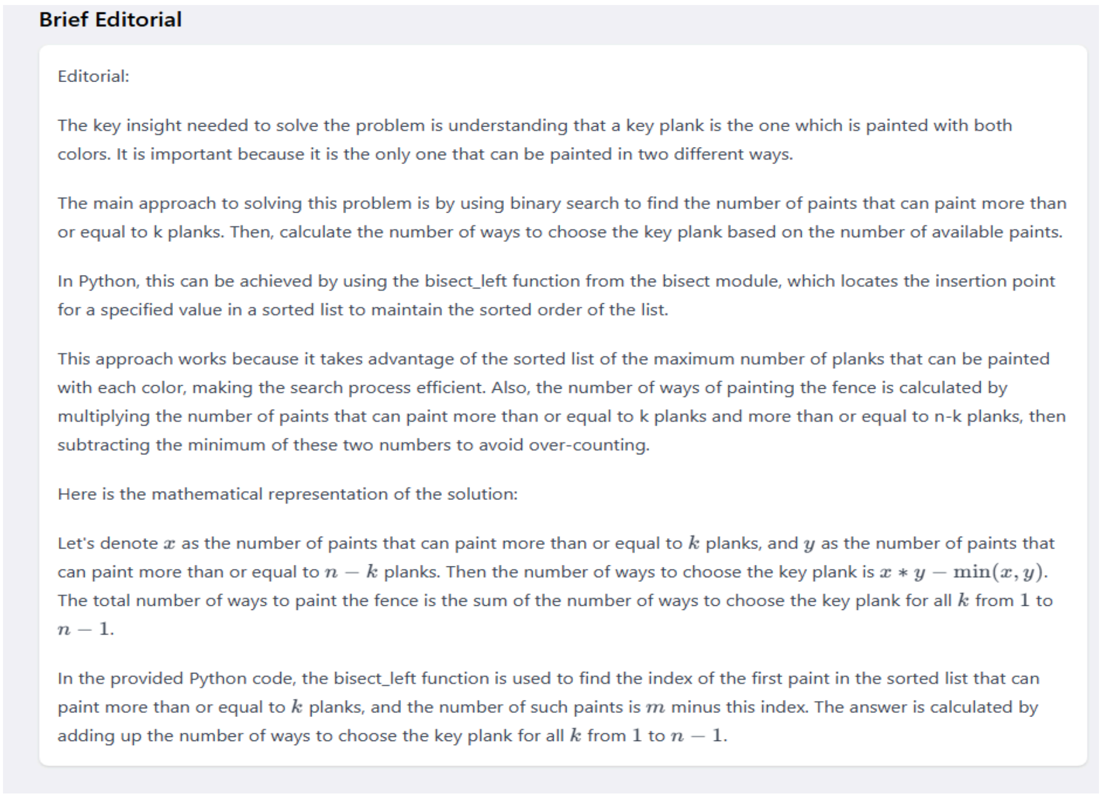
Hint 1: The problem requires you to find the number of different ways to paint a fence with planks using two different colors, under certain conditions. One approach to solve this problem is to use binary search.

Hint 2: Firstly, sort the array of available paint quantities in ascending order. Then, iterate over the possible number of planks that can be painted with one color (from 1 to n-1).

Hint 3: For each possible number of planks `k` that can be painted with one color, calculate the number of paints `x` and `y` that can paint `k` and `n-k` planks respectively. This can be done using binary search on the sorted array of available paint quantities.

Hint 4: Finally, add to the answer the product `x*y` minus the minimum of `x` and `y`. This is because the minimum of `x` and `y` is the number of ways in which the planks can be painted in the same color sequence, which are counted twice in `x*y`.

We can observe different outcomes generated, depending upon the modes: Hints, Brief Editorial, Detailed Editorial.



Brief Editorial

Editorial:

The key insight needed to solve the problem is understanding that a key plank is the one which is painted with both colors. It is important because it is the only one that can be painted in two different ways.

The main approach to solving this problem is by using binary search to find the number of paints that can paint more than or equal to k planks. Then, calculate the number of ways to choose the key plank based on the number of available paints.

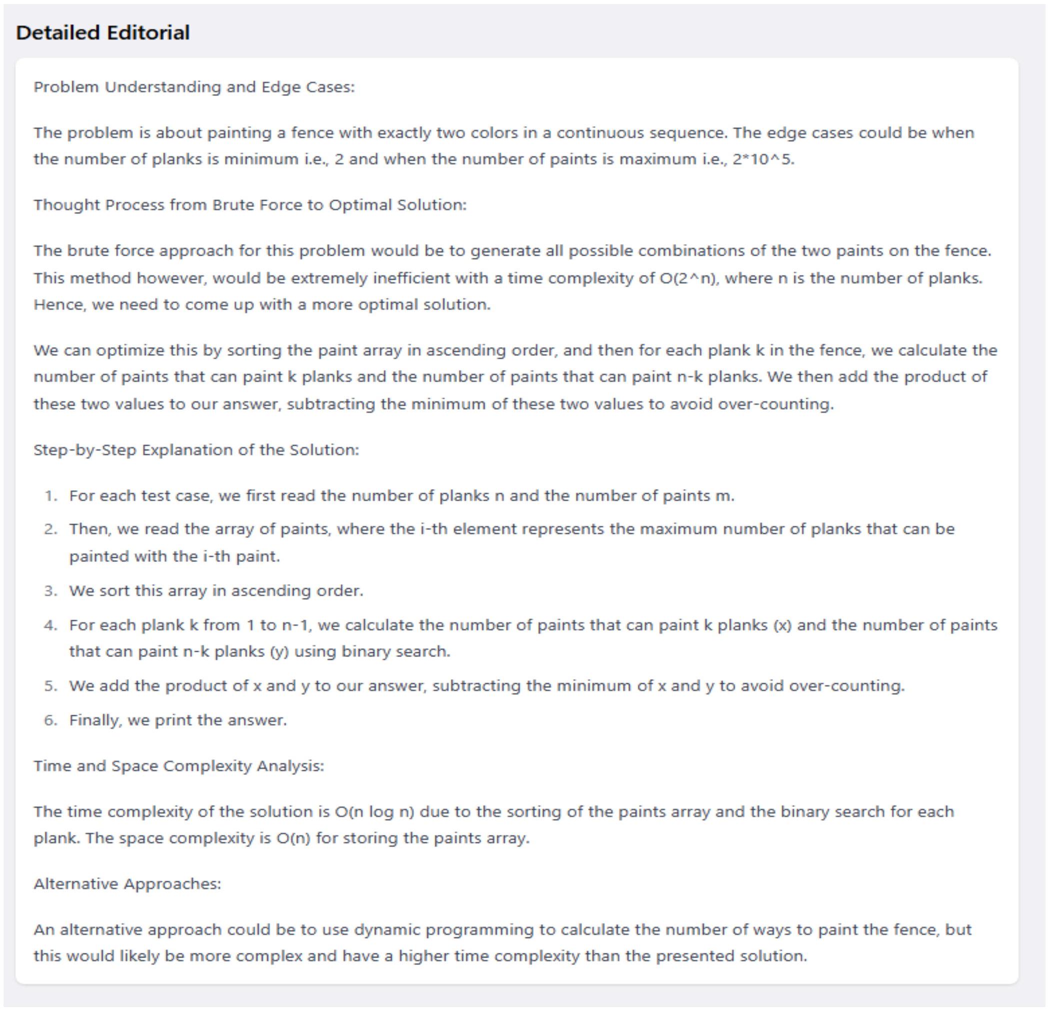
In Python, this can be achieved by using the bisect_left function from the bisect module, which locates the insertion point for a specified value in a sorted list to maintain the sorted order of the list.

This approach works because it takes advantage of the sorted list of the maximum number of planks that can be painted with each color, making the search process efficient. Also, the number of ways of painting the fence is calculated by multiplying the number of paints that can paint more than or equal to k planks and more than or equal to n-k planks, then subtracting the minimum of these two numbers to avoid over-counting.

Here is the mathematical representation of the solution:

Let's denote x as the number of paints that can paint more than or equal to k planks, and y as the number of paints that can paint more than or equal to $n-k$ planks. Then the number of ways to choose the key plank is $x * y - \min(x, y)$. The total number of ways to paint the fence is the sum of the number of ways to choose the key plank for all k from 1 to $n-1$.

In the provided Python code, the bisect_left function is used to find the index of the first paint in the sorted list that can paint more than or equal to k planks, and the number of such paints is m minus this index. The answer is calculated by adding up the number of ways to choose the key plank for all k from 1 to $n-1$.



Detailed Editorial

Problem Understanding and Edge Cases:

The problem is about painting a fence with exactly two colors in a continuous sequence. The edge cases could be when the number of planks is minimum i.e., 2 and when the number of paints is maximum i.e., 2*10^5.

Thought Process from Brute Force to Optimal Solution:

The brute force approach for this problem would be to generate all possible combinations of the two paints on the fence. This method however, would be extremely inefficient with a time complexity of O(2^n), where n is the number of planks. Hence, we need to come up with a more optimal solution.

We can optimize this by sorting the paint array in ascending order, and then for each plank k in the fence, we calculate the number of paints that can paint k planks and the number of paints that can paint n-k planks. We then add the product of these two values to our answer, subtracting the minimum of these two values to avoid over-counting.

Step-by-Step Explanation of the Solution:

1. For each test case, we first read the number of planks n and the number of paints m.
2. Then, we read the array of paints, where the i-th element represents the maximum number of planks that can be painted with the i-th paint.
3. We sort this array in ascending order.
4. For each plank k from 1 to n-1, we calculate the number of paints that can paint k planks (x) and the number of paints that can paint n-k planks (y) using binary search.
5. We add the product of x and y to our answer, subtracting the minimum of x and y to avoid over-counting.
6. Finally, we print the answer.

Time and Space Complexity Analysis:

The time complexity of the solution is O(n log n) due to the sorting of the paints array and the binary search for each plank. The space complexity is O(n) for storing the paints array.

Alternative Approaches:

An alternative approach could be to use dynamic programming to calculate the number of ways to paint the fence, but this would likely be more complex and have a higher time complexity than the presented solution.

Appendix

Section	Description	Reference(s)
Model Selection	Rate limit comparisons	https://console.groq.com/docs/rate-limits
Model Selection	Comparison ranking chart for various llm models	https://evalplus.github.io/leaderboard.html
Prompt Engineering	Prompt Strategy Optimizer + other suggestions' basis	https://aws.amazon.com/blogs/machine-learning/evaluating-prompts-at-scale-with-prompt-management-and-prompt-flows-for-amazon-bedrock/
Project Description	To cite present omegaUp's state	https://omegaup.com/arena/problem/Formados-en-la-cafeteria/
UI/UX	Mascot name for AI bot	https://www.kattis.com/
Design Considerations	To cite AlgoUniversity's Codegenie	https://www.algouniversity.com/dashboard/
Bonus Section	Reference to my Editorial Generator built from scratch	https://taupe-pasca-cb0b34.netlify.app/