International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 05 | May 2026 www.irjet.net p-ISSN: 2395-0072

Token-Cost-Aware Scheduling Middleware for LLM-Based FastAPI Applications

Shirley Dhawadkar [1], Dr. Mrs. Pratibha Adkar [2]

[1,2]*MCA Department, P E S Modern College of Engineering, Pune, India*

Abstract- *The widespread use of Large Language Models(LLMs) in web applications has introduced cost management challenges due to token-based pricing models. Existing optimization techniques primarily focus on model selection or inference-level control, with limited attention given to cost-aware scheduling at the API layer. Conventional FastAPI-based LLM services process requests without estimating output token expansion prior to model invocation, leading to inefficient cost management under concurrent workloads. This paper proposes a token-cost-aware scheduling framework for LLM-powered API systems. The framework integrates a middleware layer that estimates output token length before inference and calculates projected cost using predefined pricing parameters. A configurable scheduling policy then determines request handling based on estimated cost and Service Level Agreement (SLA) constraints. The approach is evaluated using simulated concurrent request scenarios and compared with a baseline cost-agnostic scheduling method. Results demonstrate improved token utilization efficiency while maintaining acceptable latency. The proposed framework provides a modular and reproducible solution for cost-aware deployment of LLM services.*

Keywords - Large Language Models, Token-Based Pricing, Middleware Scheduling, FastAPI, Cost-Aware Systems, API Optimization

I. INTRODUCTION

Large Language Models (LLMs) have become foundational components in modern AI-driven applications, enabling advanced capabilities such as conversational agents, code generation, content summarization, and intelligent decision support. With the integration of LLMs into web-based systems through frameworks such as FastAPI, scalable and real-time AI services are now widely accessible. However, the operational model of these systems introduces new economic and architectural challenges.

Most commercial LLM providers adopt token-based pricing structures, where both input and generated output tokens contribute to inference cost. Under such pricing schemes, the cost of processing a request is directly proportional to the number of tokens consumed. In practical deployments, output token length can vary significantly depending on prompt complexity, reasoning depth, and user instructions. As a result, request-level costs become highly unpredictable.

Conventional API scheduling mechanisms, including FIFO and fair scheduling strategies, are cost-agnostic. They process incoming requests without considering projected token usage or estimated financial impact. In high-concurrency environments, this may lead to inefficient token allocation, cost overruns, and violations of operational constraints such as budget limits or Service Level Agreements (SLAs).

Existing research primarily focuses on reasoning optimization, model routing, and distributed infrastructure scheduling. However, limited attention has been given to middleware-level cost estimation and scheduling decisions at the API boundary prior to model invocation. There remains a gap between model-level optimization techniques and deployment-level economic efficiency in LLM-powered web services.

To address this gap, this paper proposes a token-cost-aware scheduling middleware for FastAPI-based LLM applications. The proposed system performs pre-inference token estimation, calculates projected request cost using configurable pricing parameters, and applies cost-informed scheduling policies under SLA constraints. By integrating economic awareness at the API boundary, the framework aims to enhance token utilization efficiency while maintaining acceptable latency performance.

II. LITERATURE REVIEW

The rapid adoption of Large Language Models (LLMs) in web-based systems has introduced new challenges related to cost management, scheduling, and efficient deployment. Existing research primarily focuses on reasoning optimization, distributed scheduling, and edge-level resource management. However, limited work addresses token-based cost prediction and middleware-level scheduling for API-driven LLM systems.

Ong et al. (2025)

Introduced RouteLLM, a learning-based routing framework designed to optimize cost–quality trade-offs across multiple language models. The system dynamically selects between high-capacity and smaller models using preference learning to reduce overall inference cost while maintaining output quality. Although the framework addresses cost efficiency at the model-selection level, it does not incorporate token usage prediction or request-level scheduling at the API boundary. Cost decisions are

made through model routing rather than token estimation prior to inference.

Han et al. (2024)

Proposed Token-Budget-Aware LLM Reasoning (TALE), which aims to reduce unnecessary token generation in chain-of-thought reasoning. The framework introduces mechanisms to estimate and constrain token budgets dynamically during inference. Experimental results indicate that limiting redundant reasoning steps can significantly reduce token consumption without compromising performance. However, TALE operates within the model's generation process and does not extend to middleware-level scheduling or pre-inference cost prediction in web-based deployments.

Yao et al. (2023)

Proposed React, a framework that integrates reasoning and acting in LLMs to improve interpretability and task performance. The approach enables models to generate intermediate reasoning traces before producing final outputs, enhancing accuracy in knowledge-intensive tasks. While the framework demonstrates that reasoning complexity influences token generation behavior, it does not address token-based economic cost implications or deployment-level scheduling strategies. The study primarily focuses on reasoning quality rather than inference cost optimization.

Cao et al. (2020)

Provided an overview of edge computing architectures, key technologies, and cloud-edge collaboration models. The paper compares centralized cloud systems with distributed edge architectures and identifies advantages such as reduced latency, improved security, and lower bandwidth usage. However, similar to other infrastructure-level studies, it does not address predictive cost modeling or request-level scheduling for AI-based APIs operating under usage-based pricing structures.

Mao et al. (2019)

Presented Decima, a reinforcement learning-based Scheduling framework for distributed data processing clusters. The system learns workload-specific scheduling policies using graph neural networks and reinforcement learning to optimize job completion time. Decima demonstrates that intelligent scheduling can outperform traditional heuristics such as FIFO and fair scheduling. Nevertheless, the framework targets large-scale cluster environments and focuses on performance optimization rather than economic cost estimation. It does not consider token-based pricing models or API-layer scheduling decisions in LLM-powered services.

Khan et al. (2019)

Conducted a comprehensive survey on Edge Computing paradigms, including Cloudlets, Fog Computing, and Mobile Edge Computing. The study emphasizes reducing latency by pushing computation closer to users and highlights distributed resource management strategies. While the survey discusses architectural efficiency and workload distribution, it does not explore application-layer cost modeling or token-based economic constraints in AI-driven systems.

III. RESEARCH GAP

From the reviewed literature, it is evident that prior research has addressed reasoning optimization, distributed scheduling, and resource management in cloud and edge environments.

However, limited attention has been given to middleware-level cost-aware scheduling mechanisms for LLM-powered web applications. Existing works either optimize reasoning
Efficiency, selects between models, or manages cluster resources, but none integrate token-length prediction with API-boundary scheduling under Service Level Agreement (SLA) constraints.
This gap motivates the proposed research, which introduces a lightweight token-cost-aware scheduling middleware for FastAPI-based LLM systems. The proposed framework performs pre-inference token estimation, projected cost calculation, and configurable scheduling decisions at the API boundary, thereby bridging the gap between model-level optimization and deployment-level economic efficiency

IV. SYSTEM ARCHITECTURE

The proposed system adopts a layered and modular architecture designed to integrate token-cost-aware scheduling into LLM-based FastAPI applications. The architecture is structured to ensure efficient request handling, cost-aware decision-making, and role-based system interaction. It consists of distinct layers that collectively facilitate user interaction, request processing, scheduling, and monitoring.

1. Presentation Layer

The Presentation Layer provides the user-facing interface of the system and supports role-based access for both client users and administrators. It is implemented as a web-based dashboard integrated with FastAPI services. The system includes the following interfaces:

i. Authentication Interfaces: Login and registration modules that allow users to securely access the system using API keys and credentials.

ii. User Dashboard: Enables client users to view request summaries, submit new prompts, and monitor request status.

iii. Request Management Interface: Allows users to create new requests and view historical request data along with scheduling decisions.

iv. Configuration Interface: Displays system parameters such as token pricing and concurrency limits.

v. Admin Dashboard: Provides administrators with system-wide insights including total users, request statistics, and operational controls.

vi. Reports Interface: Presents analytical insights such as request distribution and token usage across users.

vii. The Presentation Layer ensures structured interaction with the system and provides transparency in terms of request processing and cost metrics.

2. Application Logic Layer

The Application Logic Layer constitutes the core processing unit of the system. It is implemented using FastAPI and handles all backend operations, including request validation, token estimation, cost computation, and scheduling.

The key functional components include:

i. Request Processing Module: Handles incoming requests and validates user inputs

ii. Token Estimation Module: Calculates input tokens and predicts output tokens using heuristic-based estimation.

iii. Cost Calculation Engine: Computes the projected cost of each request using predefined pricing parameters.

iv. Scheduling Engine: Determines the execution priority of requests based on estimated cost and system constraints.

v. Execution Handler: Invokes the LLM API for approved requests and processes responses.

3. Database Layer

The Database Layer is responsible for persistent storage and management of system data. It is implemented using a relational database system and supports efficient data retrieval and analysis.

The primary data entities include:

i. Users: Stores user credentials, roles (client/admin), and API keys.

ii. Requests: Maintains records of all submitted prompts along with their status and scheduling decisions.

iii. Token Usage: Stores estimated and actual token consumption for each request.

iv. Cost Logs: Records predicted and actual cost associated with each request.

v. System Configuration: Maintains parameters such as token pricing, maximum cost thresholds, and concurrency limits.

vi. The relational structure ensures data consistency and enables analytical reporting for system evaluation.

4. Middleware and Scheduling Layer

The Middleware Layer represents the central contribution of the system and operates between the API endpoint and the LLM service. It introduces cost-aware intelligence into request handling. Its primary responsibilities include:

i. Intercepting incoming requests before execution

ii. Estimating token usage based on input prompts I

iii. Computing projected cost using pricing configurations

iv. Assigning priority levels (e.g., high or low)

v. Making scheduling decisions based on cost constraints

vi. Managing request queues and execution order

Additionally, the system provides an admin-controlled scheduler interface, where administrators can fetch and process pending requests. This ensures controlled execution and enables demonstration of scheduling behavior.

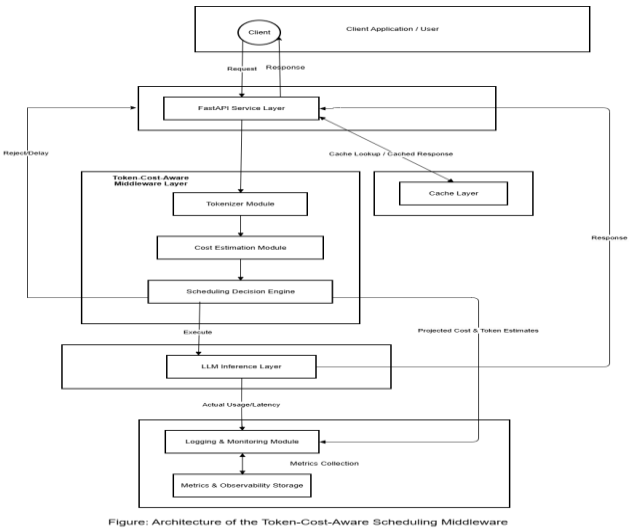


Figure: Architecture of the Token-Cost-Aware Scheduling Middleware

V. WORKING OF THE SYSTEM

The system follows a structured workflow that integrates cost estimation and scheduling prior to model execution.

1. User Authentication The process begins with user authentication through login or registration interfaces.

Each user is assigned an API key, which is used for request identification and tracking.

2. Request Submission Client users submit prompt requests through the request interface. Each request is recorded in the database with an initial status.

3. Token Estimation The system computes the number of input tokens and estimates output tokens based on predefined heuristics. This step provides an early approximation of resource consumption.
4. Cost Calculation Using token estimates and pricing parameters, the system calculates the projected cost of the request. This enables cost-aware evaluation before execution.

5. Scheduling Decision The scheduling engine assigns a priority level to the request and determines its execution status (e.g., pending or ready for processing). High-cost requests may be deprioritized, while low-cost requests may be prioritized.

6. Request Processing The administrator accesses the scheduler module to fetch pending requests and initiate processing. The system invokes the LLM API and generates responses for approved requests.

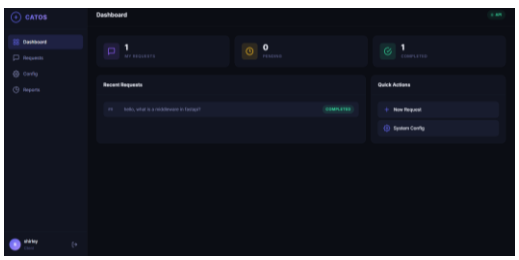
7. Logging and Storage The system records actual token usage, actual cost, and execution status. This data is stored for performance evaluation and reporting.

8. Response Delivery The generated response is returned to the user and displayed in the dashboard along with associated metrics.

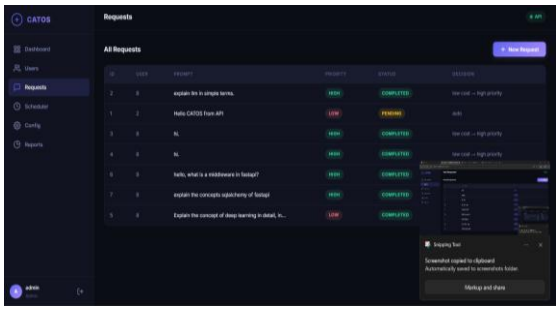
VI. SYSTEM SCREEENS

1. Login and Registration Screens: Enable secure access to the system.

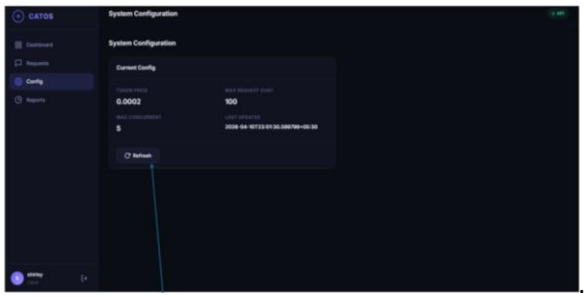
2. User Dashboard: Displays request statistics and recent activity.



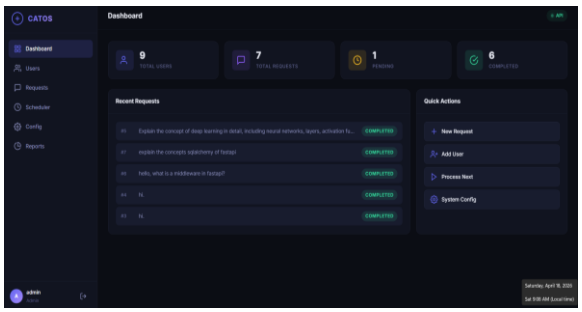
3. Request Interface: Allows users to submit prompts and view request history.



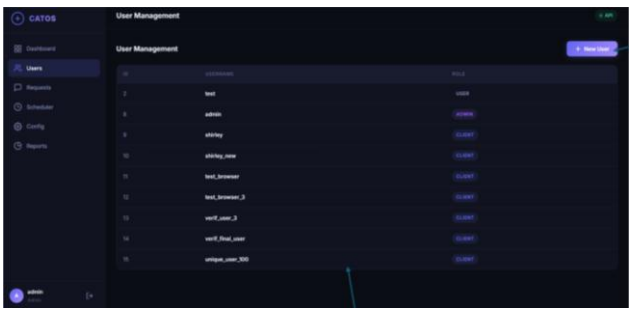
4. Configuration Screen: Displays current System parameters



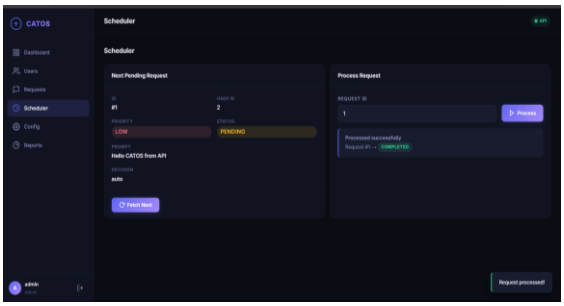
5. Admin Dashboard: Provides system-wide Insights and controls.



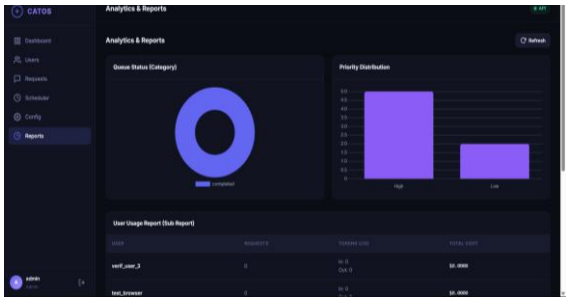
6. User Management Screen: Enables Administrators to manage users and roles.



7. Scheduler Interface: Allows administrators to fetch and process pending requests.



8. Reports Interface: Displays analytical Insights such as request distribution and token Usage.



These interfaces collectively provide a comprehensive view of system operations and enhance usability

VII. APPLICATIONS

1. AI-powered web applications for automated content generation
2. Enterprise systems requiring cost-controlled API usage
3. SaaS platforms offering LLM-based services
4. Educational systems utilizing AI tools
5. Cloud-based deployments requiring scalable request handling

VIII. ADVANTAGES

1. Enables cost-aware request processing through token estimation
2. Improves resource utilization via intelligent scheduling
3. Provides transparency through detailed logging and reporting
4. Supports role-based system interaction and control
5. Offers modular and scalable architecture
6. Maintains simplicity while supporting production-level features

IX. LIMITATIONS

1. Token estimation accuracy may vary depending on prompt complexity
2. System depends on external LLM APIs for response generation
3. Scheduling decisions are rule-based and not adaptive
4. Manual intervention is required for request processing in the current implementation
5. Limited real-time automation in scheduling and execution

X. CONCLUSION

The proposed system introduces a cost-aware middleware framework for managing LLM-based API requests within FastAPI applications. By incorporating token estimation and cost prediction prior to execution, the system enables informed scheduling decisions that optimize resource utilization and reduce operational costs.

The architecture effectively integrates user interaction, middleware processing, and administrative control into a unified system. The inclusion of role-based dashboards, scheduling mechanisms, and reporting tools enhances both usability and system transparency.

The implementation demonstrates that cost-aware scheduling at the API boundary is both feasible and beneficial for managing LLM workloads. While the current system employs rule-based scheduling, future enhancements automated scheduling strategies, adaptive learning mechanisms, and real-time cost monitoring.

Overall, the system provides a practical, scalable, and production-oriented approach for optimizing LLM-based services in modern applications.

EXAMPLE FOR REFERANCES

1) Ong, A. Almahairi, V. Wu, W.-L. Chiang,T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica, “RouteLLM: Learning to RouteLLMs with Preference Data,” in Proceedings of the International Conference on Learning Representations (ICLR), vol. 1, no. 1, pp. 1–15, 2025.
2) T. Han, Z. Wang, C. Fang, S. Zhao, S. Ma, and Z. Chen, “Token-Budget-Aware LLM Reasoning,” in Findings of the Association for Computational Linguistics (ACL Findings), vol. 2024, no. 1, pp. 1–14, 2024.
3) S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, “ReAct: Synergizing Reasoning and Acting in Language Models,” in Proceedings of the International Conference on

Learning Representations (ICLR), vol. 1, no. 1, pp. 1–21, 2023.

4) K. Cao, Y. Liu, G. Meng, and Q. Sun, "An Overview on Edge Computing Research," IEEE Access, vol. 8, no. 1, pp. 85714–85728, 2020.

5) W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge Computing: A Survey," Future eneration Computer Systems, vol. 97, no. 1, pp. 219–235, 2019.

6) H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Learning Scheduling Algorithms for Data Processing Clusters," in Proceedings of the ACM SIGCOMM Conference, vol. 49, no. 4, pp. 270–288, 2019