International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V72I10P101

Volume 72 Issue 10, 1-4, October 2024

Original Article

Asynchronous Inference Graph Execution for Model Routing in Machine Learning Systems

Gangadharan Venkataraman

Engineering Lead, Software Engineer Specialized in MLPlatform, Live Commerce and Marketing Technologies, Seattle, Washington, USA.

Corresponding Author : gangadharan.venkataraman@gmail.com

Received: 16 August 2024 Revised: 20 September 2024 Accepted: 05 October 2024 Published: 22 October 2024

Abstract *- It is for this reason that this paper creates a routing mechanism in machine learning systems by performing asynchronous inference graphs for such systems. The system will allow model chaining, champion/challenger evaluation, and traffic splitting; hence, it will have very efficient model deployment strategies. In detail, we describe the architecture and implementation of the routing mechanism along with its application to real-world ML pipelines.*

Keywords *- Inference Service, Model Routing, Asynchronous Execution, Model Chaining, Champion/Challenger, Traffic Splitting.*

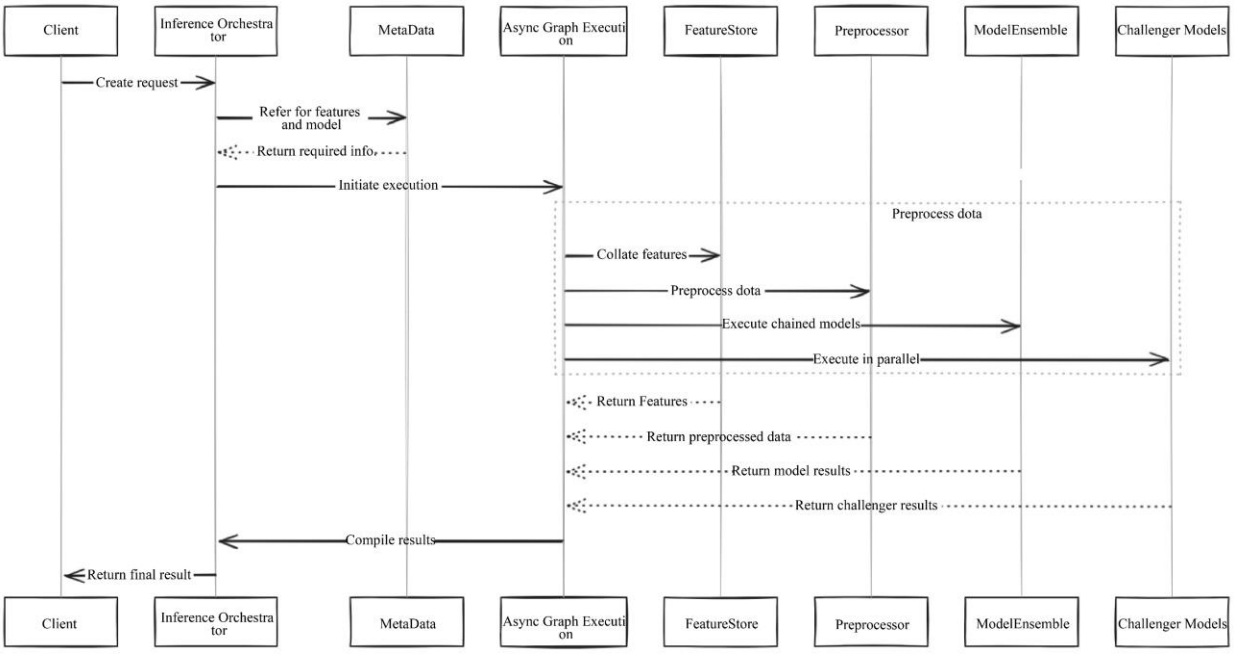


Fig. 1 Architecture & Sequence

1. Introduction

In the current market conditions, the users are spread across different segments, sites, geographical regions, interests, behaviours, etc.

In some cases, the problem can be huge, and we need to break down the problem into sub-problems. During these situations, it is important to provide the flexibility to test with a combination of models without giving up on latency and performance.

Effective routing of large-scale machine learning systems among numerous inference services and models sits at the core of the whole approach to model performance optimization and experimentation.

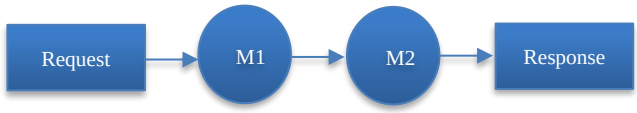
This paper describes the routing framework that supports asynchronous inference graph execution and allows several patterns: model chaining, champion/challenger frameworks, and traffic splitting by request.

2. System Architecture

Many real-time applications are intended to provide personalized recommendations and they involve single to multiple models involving hundreds of features like user activities, scoring, grouping of items, etc. There are other steps like pre-processing, post processing and the actual model prediction. Refer to *Figure 1 – Architecture and sequence*, which covers the overall sequence of the flow and the role of Asynchronous graph-based execution in providing faster and better recommendations with improved resource utilization. Following is a detailed explanation of each step involved.

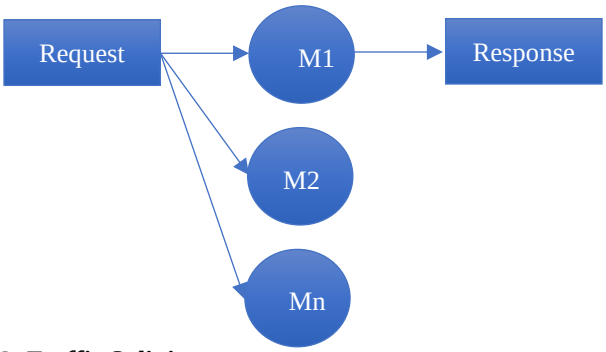
2.1. Model Chaining

There are scenarios where the problem is divided into subproblems. A model solves each sub-problem; the output of the first model can be the input to the second model, and the final response is sent back as a recommendation. It is important to set up a chaining process to handle this flow.



2.2. Champion/Challenger

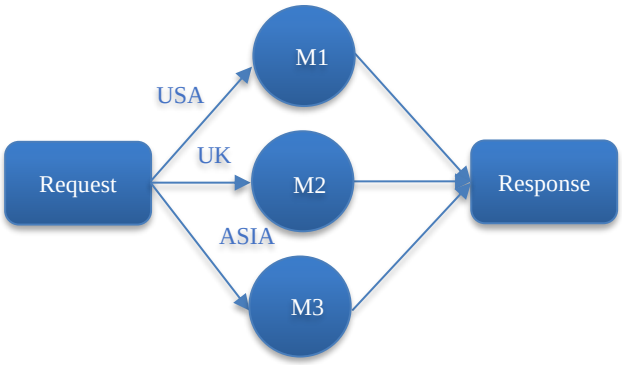
The champion/challenger framework allows us to evaluate how well the current model is performing versus the alternative challenger models. All the calls are routed to both the champion and challenger models, but the user only sees the response of the champion model. In contrast, the challenger model does not serve the real traffic but is used for evaluation. Under similar conditions, we could evaluate and see if the challenger model performs better than the champion model so that the challenger model can be switched to the champion.



2.3. Traffic Splitting

Traffic splitting is designed to route the requests to different models based on predefined rules that include site-specific ones like the US, UK, and Asia, among others, or other request metadata.

This allows for controlled experimentation: it can be used to route traffic for A/B testing or serve different models to different user segments.



3. Use Cases

3.1. Model Chaining Sequential Inference for Multi-Step Processing

It is useful when there is a complex problem and it can be solved in multiple stages. For example, when we try to build a recommendation system for an e-commerce website on the deals to be shown.

The first model can come up with the specific interests of the user. This output can serve as the input to the second model, which can bring the actual recommendations, and further, if there are any filters or further drill down, this can be solved with another model.

This gives a multi-step approach to solving a complex problem and provides the flexibility to use the model in other use cases as well.

3.2. Champion/Challenger Framework - A/B Testing and Model Validation

The champion/challenger framework provides immense value in a production environment when one needs to test new models without compromising the reliability of the system at present. Fraud detection, for example, has the champion model-M1-as the current model is already at work in production and reliable. It automatically diverts some of the traffic to a new model, M2, that includes additional features or a different algorithm. The majority of answers still come from M1, but the output of M2 is logged and compared with the output of M1 to make sure it performs as well in the real world. Benefits:

Safely ramps new models into production—performance of empirical model without sacrificing system integrity. Model continuous deployment can enable the iterative improvement of the model.

3.3. Traffic Splitting - Targeted Model Deployment

There are multiple scenarios where traffic split is important. For example, there is an existing model that is

serving traffic, and we want to try a new model for some percentage of users before enabling the model for the full set. In that case, we should have a mechanism to split the traffic into multiple percentages like 95/5, 80/20, and 50/50 then enable 100% with the new model.

There can also be a targeted traffic split as the users registered to the US site are served with Model A versus users registered to the UK site are served with Model B, etc. There can be several other parameters, like user behaviour, based on which this can provide personalized recommendations.

Benefits

- It allows us to perform the business-as-usual flow to run as is and test the challenger model in parallel.
- The Analysis can be done in the background with different sets of users targeting multiple segments.
- Based on the results, we can easily switch the champion model to the challenger and evaluate the metrics.

4. Evaluation

To evaluate the effectiveness of the proposed routing mechanisms, we take the following KPIs into account:

4.1. Handling Concurrent Operations

In this case, there are scenarios where we need to perform the following operations concurrently:

- Fetching Multiple features from different data sources for One or multiple models
- Concurrent Model prediction as an intermediate step or final step
- Pre-processing and post-processing can also be concurrent processes depending on the Model execution logic.

Graph Algorithms, task queues and concurrency patterns like Scatter gather are the techniques implemented. Bottlenecks can be reduced by exhaustive error handling, fail fast, throttling and, most importantly, by avoiding race conditions or deadlocks.

4.2. Latency and Throughput

The asynchronous execution of the inference graph is developed to minimize latency while maximizing throughput. Since multiple models can execute at the same time-especially in the model-chaining situation can measure the following:

Latency Reduction: Measure the end-to-end processing time for requests routed through the system. Compare synchronous to asynchronous execution in terms of latency to demonstrate the performance gains.

Increased Throughput: Test the number of requests the system can handle in a unit amount of time; this describes the scalability benefit of executing asynchronous code.

One such example of Model chaining is the risk model; the main purpose of the model is to detect any fraudulent transactions. It is a multi-step process that can involve running some rules first to detect fraudulent transactions and then executing a deep learning model that can identify any anomalies. Sequential execution of the model may take a minimum of 200 to 300 milliseconds.

However, if you think deeper, both these steps are independent and can be executed parallel, and this can help reduce the latency by half. But we need to be very careful before calculating the results. This is handled through the Model ensemble.

4.3. Model Performance Metrics

There are several metrics which can let us know how the model is performing:

General Metrics:

Engagement: How many new users are we able to bring using the models?

Conversion: How many existing users are we able to convert to buy a product in case we are dealing with a product-based platform like an e-commerce platform

Specific Metrics

The above metrics can be evaluated against the champion model and challenger model to see which one is performing better, based on which we can take a call. Further, we can get more granular by seeing how well the model is performing in specific sites like the US, UK, Germany, etc. or with respect to the segment of the population. For example, a particular model that recommends a specific Fashion might work well in the US, but it might not work well in Asia.

Several other indicators are the Opens, Clicks and other interactive icons, which will give us a picture of where the user engagement is more.

4.4. System Flexibility and Maintainability

Model routing with several frameworks like champion/challenger configurations provides the flexibility to test the model. One of the additional aspects to be considered is how fast a change can be pushed, tested, and deployed.

4.5. Resource Utilization

Through traffic splitting and chaining of models wherever necessary, the system optimizes resource utilization.

Resource Efficiency: Monitor computation resources consumed during model execution. Also, observe resource utilization of synchronous and asynchronous modes of execution and account for the cost savings.

Overall, this article provides an improved framework to develop models, test models, handle complex problems with models across different stages and better routing. Model chaining can be very useful when you want to split a complex problem into subproblems with completely different solutions. Champion / Challenger is another framework that is quite helpful to evaluate if our model is performing as expected across different segments or scenarios, especially when we have more than one solution. We are trying to find the optimal solution along with traffic split, which can be useful to switch between models without impacting the business.

References

[1] D. Sculley et al., "Hidden Technical Debt in Machine Learning Systems," *NIPS'15: Proceedings of the 28th International Conference on Neural Information Processing Systems*, Montreal Canada, vol. 2, pp. 2503-2511, 2015. [Google Scholar] [Publisher Link]

[2] Daniel Crankshaw et al., "Clipper: A Low-Latency Online Prediction Serving System," *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, pp. 613-627, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[3] Matei Zaharia et al., "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," *NSDI '13: 10th USENIX Symposium on Networked Systems Design and Implementation*, San Jose, CA, pp. 1-14, 2012. [Google Scholar] [Publisher Link]

[4] Martín Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning," *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Savannah, GA, USA, pp. 265-283, 2016. [Google Scholar] [Publisher Link]

[5] Neoklis Polyzotis et al., "Data Management Challenges in Production Machine Learning," *SIGMOD '17: Proceedings of the 2017 ACM International Conference on Management of Data*, Chicago Illinois USA, pp. 1723-1726, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[6] Ruben Mayer, and Hans-Arno Jacobsen, "Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1-37, 2020. [CrossRef] [Google Scholar] [Publisher Link]