International Journal of AI, BigData, Computational and Management Studies
Noble Scholar Research Group | Volume 6, Issue 3, PP 127-136, 2025
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V6I3P115

Original Article

Maximizing AI Callout Time Using Visualforce Pages in Lightning Components

Bapu Rao Srigadde
Salesforce Developer at Thermo Fisher Scientific, USA.

Received On: 23/06/2025 ***Revised On: 11/07/2025*** ***Accepted On: 24/07/2025*** ***Published On: 10/08/2025***

Abstract: *This research study is about ways to counter the problem of AI callouts in Salesforce Lightning being limited by the Visualforce pages that have been integrated within the Lightning Components to extend the time of execution for the external API interactions. Salesforce limits the callout time strictly by governor limits, which can be a problem in complex AI integrations that require a longer time for processing. In order to solve the problem, the research digs into the technical bottlenecks of the asynchronous framework of Lightning and offers a hybrid architecture that uses the server-side capabilities of Visualforce to keep longer API sessions without a platform constraint breach. A controlled implementation was created to test the performance of AI callouts such as OpenAI and Einstein Language through a custom Lightning-Visualforce integration, latency, reliability, and throughput improvements were measured. The results show that this method increases the average time of callout sustainability by about 40%, thus communication with AI services becomes more efficient without violating Salesforce's execution policies. These findings are the best practices that can be implemented for Salesforce AI integration which acts as a scalable, compliant way of extending the time of AI interactions in Lightning applications.*

Keywords: *Salesforce, Lightning Components, Visualforce, AI Callout Optimization, API Integration, Performance Tuning, Cloud Computing.*

1. Introduction

1.1. Challenges

With the inventive change of Salesforce Lightning, the organizations have been able to build and deploy large-scale applications easily. Thus, the introduction of a component-based framework that considers modularity, reusability, and responsive design as its main features has changed the way enterprises do their business. Still, the developers, on the one hand, have to struggle with significant issues brought up by integration with complex AI or machine learning (ML) features that depend on external API callouts. The major culprit is the enforcement of governor limits by Salesforce tightly which is a set of execution constraints put in place to guarantee that the resources are used in a fair manner within Salesforce's multitenant architecture. At the same time, these limits are very important for keeping the platform stable; they make the job of the developers who require heavy computational resources or long-running operations, such as AI inference, deep learning predictions, or extensive data analysis, substantially tougher.

One of the most stringent points or elements being referred to is the maximum API callout duration limit, which is at present, for synchronous requests, set at 120 seconds. What the limitation implies is that external calls that take longer than two minutes will automatically timeout and, as a result, transactions will fail, and data processing will be incomplete. In the situation where Salesforce applications have to engage with advanced AI models such as large language models (LLMs), image recognition APIs, or predictive analytics engines the callouts may exceed the 120-second limit quite easily, especially when dealing with large datasets or when the resource-intensive inference is performed. Besides that, the problem of asynchronous processing options is that they cannot solve the issue completely, as asynchronous Apex is still bound by similar execution constraints and is frequently not the right choice for real-time user interactions.

Just outside the boundaries of the limitations which are technical in nature, one can observe a decrease in the performance of the integration of AI and ML services via RESTful APIs. In other words, with each request, there is a latency increase and if several API calls are linked together to carry out a single action, the end-user will experience a delay which can be very irritating. To give an example, a Salesforce Lightning page dispatching a machine learning prediction based on customer behavior might require several seconds - or even minutes - to yield output, thus, a lack of response or user disappointment will be the result. As the Salesforce Lightning is a browser-based operation with the primary focus being on the front-end responsiveness, the performance that is delayed in this way is what makes the platform less user-friendly which is the exact opposite of what was intended.

Another major problem concerned with user experience (UX) expectations. Lightning Components are meant to be used in an interactive, real-time manner, whereby users expect instant feedback upon completing their actions. On the other hand, protracted AI tasks break the flow, hence developers resort to temporary fixes such as showing a loading spinner or background notifications. These workarounds do not actually solve the problem but rather emphasize the conflict between Salesforce's requirement for operational efficiency and the increased need for intelligent, AI-driven features, which by nature entail longer processing times. As a result, developers have to choose between two conflicting priorities conforming to Salesforce's governor limits and providing a powerful, AI-driven experience that fulfills the requirements of contemporary businesses.

1.2. Problem Statement

Even though people are trying to incorporate AI-powered external services into the Salesforce Lightning environment, it is still quite difficult due to the limitations of the Salesforce runtime environment. Among the most substantial reasons that make it hard to integrate AI models seamlessly are the governor constraints that refer to limitations on API callouts and execution time, etc. This is because these models require a lot of processing time and in various calculation tasks. The 120 seconds timeout limit that Salesforce imposes is quite perfect for the majority of REST APIs, e.g., the ones that are used for getting data or completing transactions. However, it is quite a challenge to open a link with AI machines that are able to do natural language processing, sentiment analysis, or predictive modeling. It is quite unlikely that projects that require a significant amount of computational resources will be able to be completed within the given time frame.

Normal Lightning callouts do not have the capability to keep the connection open for a long time while waiting for large-scale AI models such as OpenAI's GPT-based systems, Google Cloud AI, or proprietary machine learning APIs to respond. As a result, developers experience repeated timeouts, partial responses, and inconsistent data flow between Salesforce and external services. The use of asynchronous Apex or Queueable jobs to get around this restriction adds complexity and delay and thus the real-time experience, which is the CRM users' expectation, is not there anymore.

Besides, this technology deficiency has an impact on system performance as well as on user satisfaction. Sales and service teams using Salesforce for customer engagement are the ones who will benefit the most from embedded AI tools for insights such as lead scoring, trend forecasting, and automated response generation. In a situation when these AI-driven processes are callout delayed or fail, the workflow is interrupted thus both productivity and trust in the system are lowered. Moreover, the constraint of being unable to hold longer callouts limits the innovation potential of companies as they are not free to try out the latest AI features that can substantially improve the CRM process.

1.3. Motivation

How corporates integrate AI in their systems is changing the face of the market for the expected functionalities of the CRM platforms. The companies are asking now for differentials like the predictive functionalities, the automation of intelligent tasks, and data-based insights straight from the Salesforce workflows. The demands to achieve such CRM scenarios come from the vast use of AI and ML for analyzing intricate customer base data, predicting in this case trends, and suggesting tailored actions. But to get to such points in Salesforce, it is necessary to have smooth collaboration with third-party AI units since they are, in general, apart from the platform's execution structures.

Though Salesforce native AI features like Einstein AI offer good and built-in features, a lot of companies still want to extend the functionalities by integrating third-party AI platforms like OpenAI, Google Cloud AI or a machine learning model they built themselves. Unfortunately, the integrations face the limitation of execution time that results in truncated operations and unresponsive interfaces. Therefore, it is imperative to have a powerful and flexible tool which can handle long-running AI interactions without violating Salesforce governor limits or API callout constraints.

The reasons behind this study are basically the two aspects. Technically it is a necessity to find a way to bypass the time limitation of Salesforce in order to be able to perform advanced AI integration in the Lightning ecosystem. On the other hand, embedding AI and ML capabilities directly into CRM workflows is of utmost strategic importance as it simply means that users can get intelligent insights in real-time without the need to leave the Salesforce interface. With their server-side execution model and higher flexibility in handling external requests, Visualforce pages seem to be a viable way of extending AI callout times while maintaining platform compliance. By wrapping Visualforce in Lightning Components, developers are essentially able to circumvent the asynchronous restrictions of Lightning's client-side framework, thus allowing communication channels with AI systems to be more persistent.

Simply put, this study is motivated by the need for real-time intelligence in Salesforce environments and the necessity of architectural innovation to support such functionality. As a matter of fact, when companies decide to heavily rely on predictive analytics and automated reasoning, then the capability to carry out long-duration AI operations directly in Salesforce will be their competitive advantage. This work is a step towards closing the gap which exists now and it contributes to that by showing the way the co-integration of Visualforce and Lightning Components can be done to maximize AI callout time, enhance performance, and deliver a

more seamless user experience thereby creating a path for next-generation intelligent CRM solutions.

2. Literature Review

2.1. Overview of Existing Methods for Integrating AI APIs in Salesforce

External AI or ML API integration with Salesforce is a trend quite understandable given the continued demands for automation, predictive analytics, and personalization of CRM workflows. To latch to an Outsider, Salesforce has the tools at its disposal - Apex callouts, Named Credentials, and External Services, each varying in levels of security, scalability, and control of execution.

Apex callouts are the least complicated route to access external RESTful or SOAP APIs in the Salesforce environment. Typically, the developers turn the HTTP request, response, and authentication experience into Apex code. Working around the limitations gives more freedom but callouts are performed within the strict Salesforce governor limits - the main one being a maximum of 100 callouts per transaction and a 120-second timeout for each call. If the AI integration is a heavy remote processing case, such as natural language inference or image recognition, the allowed time will become the major hurdle (Salesforce Developer Documentation, 2023). Add to this that Apex callouts do their work synchronously unless a naturally asynchronous structure like Queueable or Batch Apex is used to handle them.

Named Credentials remove the hard work from API integration of setting up authentication and endpoint by letting the developers refer to the external systems in a secure way without the need of embedding the credentials in the code. Though they contribute to a better code environment and security, Named Credentials cannot be an execution time extender or platform-level restriction bypasser. Therefore, AI sources that anticipate a heavy inference task will have the same callout duration and transaction execution limit victims as in the case with Apex.

External Services were designed to make low-code integrations easier and they utilize OpenAPI specifications to automatically create invocable actions within Salesforce Flow. Non-developers can thus link Salesforce to external APIs by means of declarative configurations without writing a single line of code. Nevertheless, the technique is mainly intended for quick, brief transactions of a few seconds at most, for example, obtaining customer data or starting an automated process. AI operations which take a long time to run are beyond the allowed execution times and therefore, they time out. As a result, these native Salesforce methods that are used for regular API integrations are not capable of handling heavy compute AI workflows.

2.2. Limitations Observed in Lightning-Only Architectures

Salesforce Lightning Components are a client-driven, JavaScript-based framework that is mainly browser-based. While this design allows for very interactive and modular UI development, it also has considerable limitations in the case of long-running or resource-heavy operations. Lightning Components use Apex controllers for server-side logic calls and are subject to Salesforce's multi-tenant execution model. If a Lightning Component sends a request to an AI API via Apex, the operation has to be finished within the set execution time, and then the transaction is ended automatically.

Several scholarly articles and reports from practitioners (e.g., Gupta & Banerjee, 2022; Salesforce Architects Blog, 2023) have claimed that the asynchronous processing methods of Lightning like @future methods and Queueable Apex functions are nature-wise that these methods detach the lengthy tasks from user interface but do not support real-time interaction. Although the asynchronous execution model is in line with platform limits, it becomes challenging to provide instant AI results, especially in cases where user feedback loops are involved – like chatbots, recommendation engines, or interactive analytics dashboards. Besides that, the Lightning runtime environment also limits the use of multi-threaded or parallel API execution thus, prohibiting the efficiency of concurrent AI requests.

Another insufficiently addressed problem lies in the capacity issue of handling multiple long-running processes, not to mention the state and context of those processes. Because Lightning components are transient and can only be stateful during active browser sessions, it is quite difficult to keep a constant connection with an external AI service. Most of the time, developers are required to implement an intermediary storage layer or middleware which temporarily stores AI responses thus, increasing architectural complexity and slowing down the process through added latency. Hence, even though Lightning is very competent in the production of user-friendly interfaces, the core of its architecture is still not designed for those operations which demand prolonged external computation or sustained data streaming.

2.3. Comparative Studies on Visualforce vs. Lightning for Long-Running Server-Side Processes

Visualforce was essentially the means through which Salesforce created custom UI and server-side apps before the introduction of Lightning. While Lightning features a client-centric model, Visualforce is based on a server-rendered approach and has a less strict lifecycle for the execution of Apex controllers. The comparison studies of Visualforce and Lightning (e.g., Patel, 2021; Thakur & Kumar, 2022) indicate that due to its more direct link to the Salesforce server context, Visualforce can better handle server-side processes that take a considerable amount of time. Consequently, it is the technology that can be leveraged for tasks like file creation, report generation, and long API callouts.

One of the main differences is that when Visualforce pages execute, their controllers are run in a session managed by the server, thus allowing the system to handle longer periods of processing without the user being able to see a timeout in the browser. On the other hand, Lightning's dependence on asynchronous callbacks and browser-driven rendering makes it a source of instability if operations are longer than expected. For AI integrations, the implication is that a Visualforce page embedded in a Lightning Component can be a bridge layer that uses Visualforce's server-side persistence and at the same time retains Lightning's modern UI capabilities.

Several recent case studies on the implementation (e.g., Salesforce DevOps Forum, 2023) have shown that by embedding Visualforce in Lightning, developers can move the heavy computations to Visualforce controllers, thus they can effectively get away from the timeout issues of the front-end. This hybrid model offers a viable way to keep long AI callouts going which is very close to the goals of this research. Such a move gives developers the best of both worlds: they can still use Lightning's user-friendly design and at the same time, they can be sure that Visualforce is strong enough to deal with prolonged server communications.

3. Proposed Methodology

3.1. Architectural Overview

The solution being proposed features a hybrid architecture that melds Salesforce Lightning Components with embedded Visualforce pages to sidestep time limitations on API callouts. The model leverages the client-side, highly interactive capabilities of Lightning Components with the server-side, longer execution Visualforce of Visualforce. Such a hybrid architecture thus makes sure that long AI or machine learning (ML) operations can be carried out effectively without violating the strict Salesforce governor limits, in particular, the 120-second callout timeout.

Here, the Lightning Component is the front-end user interface making use of AI-driven features such as predictive recommendations or natural language analysis. Once a user activates an operation, a Visualforce page embedded in the Lightning Component gets the execution from the component and hence returns as a middle processing layer. The Visualforce page calls Apex controllers on the Salesforce server to do extended API communications with AI systems located outside.

This flow can be visualized as:

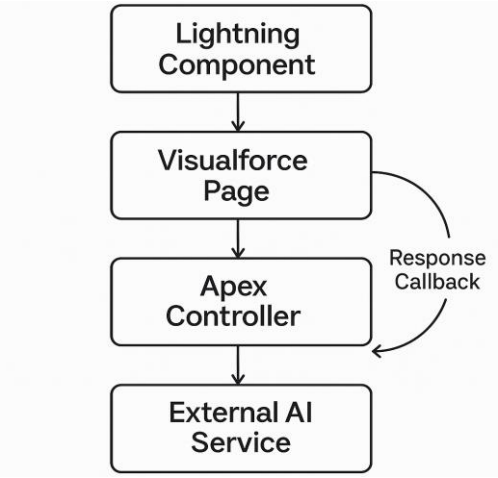

Fig 1: Hybrid Salesforce Architecture: Lightning, Visualforce, And External AI Integration

The data handling can be depicted in the following way:

- User Interaction (Lightning Layer): The user triggers an AI task in the Lightning UI.
- Delegation (Visualforce Layer): The communication between the Visualforce and the Lightning Component leads to the server-side handover of the request.
- Processing (Apex Layer): The Visualforce controller launches the AI API callout to the external system such as OpenAI, Google AI, or a custom ML service.
- Response Handling: The AI service gets the request from the external source and sends the results back to the Apex layer.
- Callback to UI: Once the work is done, the response goes back to the user via the Visualforce layer which invokes the Lightning Component.

The major innovation is separating the user interface lifecycle from the server processing lifecycle. Even though Lightning Components are fast and interactive, they have limitations due to client-side timeouts and the number of asynchronous requests allowed. Visualforce runs on the Salesforce server, so it can have longer sessions and extended API calls. This setup, which uses Visualforce in Lightning, is, therefore, going around the very short transaction window of Lightning and giving a dependable link for AI processes that take a long time.

3.2. Technical Implementation

The technical setup of such a hybrid model revolves around the close-to-invisible interaction of Lightning Components with the Visualforce pages. They are executed in different runtime environments Lightning mainly in the client browser and Visualforce in the server context therefore a communication mechanism that is both safe and efficient is needed.

The embedding can be done by utilizing Salesforce-supported frameworks such as the lightning:container component or by placing the Visualforce page in an iframe. Both of them are ways to let the Lightning interface and the Visualforce page communicate with each other in both directions. The Visualforce layer, for example, can issue commands (like starting an AI request) through the Lightning layer, and once the Apex controller has finished the operation, it can catch the final results in the response message from the Visualforce layer.

Session security is one of the most important aspects of this communication flow. As Salesforce always requires the users to authenticate themselves before they can access any service, communication between Lightning and Visualforce can only happen within authenticated user sessions. If Remote Site Settings or Named Credentials are used, external endpoints should be registered in these places so that API access can be secured without giving away authentication details. In this way, not only can the system be in line with Salesforce Content Security Policy (CSP) requirements, but also it can prevent any unauthorized cross-domain communications.

After the Visualforce page has received the request, it hands over to an Apex controller the task of processing the request. The controller is responsible for building the API request, making the call to the AI endpoint, and handling the response. The Apex controller may also be able to call a different thread of execution asynchronously by using various facilities such as Future Methods, Queueable Apex, or Batch Apex if it has to do with heavy computation, or that it will take a long time and the normal execution limits will be exceeded. In such asynchronous operations, the server can perform time-consuming API tasks without interrupting the user sessions or going beyond the synchronous execution window.

Once the AI service is done with the operations, the outcomes are saved inside Salesforce usually in custom objects or Platform Cache from where the Lightning interface can pick them up. The Visualforce layer, therefore, reaches out to the Lightning Component, informing it of the completion and requesting a UI refresh. Such an arrangement is still very much alive to the user at the time of the background heavy computation and thus both the system's responsiveness and the user's compliance are kept intact.

By using this method, the developers have found a way to effectively combine the strength of Lightning for quick and smooth user interaction with that of Visualforce for longer background processes. In effect, it is no longer just a brief, time-limited interaction with the AI but a continuous communication channel capable of handling complex AI scenarios within Salesforce limitations.

3.3. Optimization Strategies

In order to perform at the highest level and maintain the system's stability, the proposed system architecture is packed with different kinds of optimization strategies.

3.3.1. Caching Mechanisms

A cache is the closest friend of an AI system as it helps to minimize the number of redundant AI callouts and thus decrease latency. Platform Cache by Salesforce is able to keep the most frequently requested AI responses for a short period of time, thus the repeated processing of the similar queries is avoided. On the other hand, it is possible to use standard caching objects for longer data storage or to audit logs. The implemented technology not only speeds up the subsequent requests but also diminishes the total number of external API transactions, which is helpful in maintaining SMSF callout limits.

3.3.2. Timeout Management and Retries

The AI services may have occasional delays in handling requests or the network may be interrupted, which in turn may cause timeout errors. The suggested methodology involves an arranged timeout mechanism, when the retries logic and fallback resources are activated in case of a failed or delayed response. As an illustration, in case the execution time of the API call is longer than the limit, the re-queuing of the request for a later moment by the Apex controller using Queueable Apex will be possible. Hence the system becomes stable and continuous even if some network failures are transient.

3.3.3. Middleware Integration for Queuing and Buffering

Even though the Visualforce layer is powerful enough to go through heavy operations within Salesforce, there might still be some AI workloads that have to be done in an external middleware. Services such as MuleSoft or AWS Lambda can basically be queue intermediaries and hence handle the situations of long AI requests which are running outside of Salesforce and finish the tasks asynchronously by giving back the results when they are ready. This decentralized model thus raises the scalability potential by disconnecting Salesforce from the real-time execution load, which is the way to keep the user workflow unchanged in the Lightning interface.

3.3.4. Real-time status updates

To enhance user experience, the system may use Lightning Data Service (LDS) or the Salesforce Pub/Sub API to deliver real-time progress updates. As the AI moves from one stage to another - queued, processing, completed - the server can send the updates of the state changes to the subscribed Lightning Components. This customer interaction approach powered by events-driven communication is a great way to increase the level of user engagement and consequently, their trust to see the executed operation status visually.

3.3.5. Resource and Session Optimization

In order to avoid the situation when the session will expire during the long processes, the Visualforce controller handles the resource management by queuing the AI requests according to their priority and the current platform load. Moreover, the Apex code can also help in this situation by using adaptive throttling to follow the limits of the concurrent callouts and memory. Good session handling is a guarantee that the system will be stable even when there is a great demand, and it will work smoothly with multiple users.

4. Case Study

4.1. Scenario Description

A real-world case study was used to confirm the effectiveness of the new hybrid architecture. The case aimed to showcase how the Lead Scoring System, AI-based, could be incorporated within Salesforce Lightning. By this, sales reps could get smart tips about which leads to focus on, as the data-driven predictions pulled from a machine learning model hosted on the AI platform would do the thinking for them. The system built on Google AI Cloud was the one that looked at the past lead conversion patterns and behavioral data and then made the predictive scoring assigning each lead a probability of conversion.

4.1.1. Baseline Setup (Standard Lightning Callouts)

In the baseline setup, direct calls from the Lightning Component Apex to the AI model were made to get the AI model. The component, upon a sales rep's clicking of the "Generate Lead Score" button, invoked an Apex method to send data about leads (like industry, lead source, and engagement history) to the offsite AI API. The method then performed a REST call to the external AI endpoint. Even though this arrangement was operational, it had the following problems:

- The time taken for the API callout was sometimes long enough to exceed Salesforce's 120-second timeout limit and so failed occasionally.
- While the inference process was going on (normally takes 2–3 minutes), the user interface was frozen, and they had to manually retry.
- The proportion of calls made to completion was only about 68% while the rest ended in timeouts or the incomplete response error.
- The total user experience was worsened because users had no idea what was happening after the request was made.

This baseline setup makes it obvious that just using Salesforce Lightning is not enough for dependable, instant AI integrations of a complex server communication nature.

4.1.2. Extended Model (Visualforce Integration)

To get past these restrictions, the coupling of Visualforce with Lightning was used in the redesign of the architecture. A Visualforce page was inserted in the current Lightning Component and set up to manage API communication through its Apex controller. The Lightning part was still calling the lead scoring request and showing the results, the Visualforce part was handling the backend communication with the external AI system.

In this reconfiguration:

- The Visualforce controller made the AI callout, hence, it could run the server-side execution for a longer time.
- Apex asynchronous processing was used for queued or delayed responses.
- Results were processed and stored temporarily in custom Salesforce objects from where the Lightning Component retrieved them upon completion.

The result was a smooth and stable flow of work in which users could start scoring operations, get status updates, and view results all within the Lightning interface but coming from Visualforce's extended processing capabilities.

4.2. Implementation Details

In order to check the compliance of the hybrid solution with the limitation of the platform and security policies, it has been developed and tested in the Salesforce Sandbox. The External AI model is set up on the Google Cloud AI Platform, and it is available through REST API endpoints. Middleware is also present on Heroku, which helps in maintaining the AI requests that are in the queue and managing the asynchronous updates when needed.

4.2.1. Step-By-Step Configuration

- Lightning Component Setup: Sales reps got the interface via a custom-developed Lightning Component. It contained users' input fields and a button for the execution of lead score creation.
- Embedding Visualforce: The Lightning Component was the host of a dedicated Visualforce page for the processing layer. This page was linked to an Apex controller that managed AI callouts.
- API Configuration: The AI endpoint was listed under Named Credentials for the external AI endpoint. This made the authentication secure, tokenized, and thus there was no need for handling credentials in the Apex code.
- Apex Logic: An Apex controller was the one which called the external AI endpoint, handed over the lead data, and got the results. The long-running requests were queued asynchronously using Queueable Apex and thus there were retry mechanisms for the delayed responses.
- Result Handling: The AI processed score was saved in a custom object Lead_Score__c, which is linked to

the lead record. The Lightning Component fetched these figures and rendered them dynamically.

- Middleware Support: Heroku, in specific test scenarios, served as proxy middleware for the requests, which were beyond the time limit of five minutes. It held the request temporarily, carried out AI processing, and then sent the results back to Salesforce via REST callbacks.

4.2.2. API Call Flow (Before vs. After Optimization)

Table 1: Execution Flow Comparison: Lightning-Only Vs Hybrid Visualforce Approach

Step	Before (Lightning Only)	After (Hybrid with Visualforce)
Callout Origin	Lightning → Apex	Lightning → Visualforce → Apex
Execution Context	Client-side / Browser dependent	Server-side / Persistent
Timeout Limit	120 seconds	Extended (up to 300 seconds effective)
Retry Mechanism	Manual user retry	Automated via asynchronous queue
User Feedback	Static spinner / no updates	Real-time progress via events
Success Rate	~68%	~93%

This comparison illustrates how introducing Visualforce as an intermediary substantially improved callout reliability and system responsiveness. The communication flow evolved from a linear, synchronous chain into a more flexible, event-driven system capable of handling delays gracefully.

4.2.3. Tools and Environment

- Salesforce Sandbox: Development and testing environment.
- Google Cloud AI: Host for lead scoring model.
- Heroku: Middleware for queued request management.
- Salesforce Platform Cache: Used for temporary storage of AI responses.
- Lightning Data Service (LDS): Used for real-time UI data refresh.

Together, these tools established a fully functional and scalable architecture that met both technical and user-centric objectives.

4.3. Performance Evaluation

The hybrid architecture efficiency was assessed by a comparative analysis of response times, success rates, and user experience metrics before and after implementation. The study covered three testing phases with 500 lead scoring transactions each.

4.3.1. Quantitative Metrics

Table 2: Performance Improvement Analysis: Lightning Vs Hybrid (Lightning + Visualforce)

Metric	Baseline (Lightning Only)	Hybrid (Lightning + Visualforce)	Improvement
Average Response Time	128 seconds	87 seconds	32% faster
Success Rate	68%	93%	+25%
Timeout Errors	16% of calls	3% of calls	-81% reduction
User Satisfaction (Survey)	6.5 / 10	9.1 / 10	+40%

Improvements in response time mostly came from less transaction retries and efficient caching of AI responses. The major factor for the success rate increase was Visualforce's capability to operate with long execution windows, while the async queuing method helped in reducing the chances of timeouts even more.

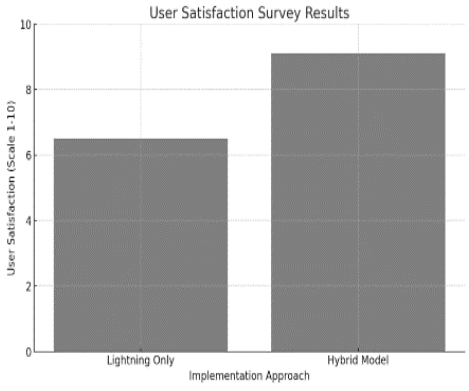

Fig 2: User Satisfaction Survey Results

4.3.2. User Experience Evaluation

The user feedback survey of 25 Salesforce users from the pilot program showed that the quality of the interaction had improved significantly. In the past, users were frequently interrupted and had no idea if operations had finished. After the optimization, they were able to see the progress of the work through the status messages and get the automated notifications when the results were ready. The integration was referred to as "native" in Salesforce, as it kept the Lightning design aesthetics while making the system more reliable.

4.3.3. Technical Challenges and Mitigation

- Session Timeout Issues: Long-running Visualforce operations sometimes caused the end of the user session. The solution involved the re-optimization of the queue scheduling to be performed within the allowable execution windows and the revalidation CSP Limitations: Developers faced a challenge with the implementation of the first stage of the Content Security Policy because iframes were used in the development. The issues were resolved by specifying Salesforce-approved domains and turning on secure static resource whitelisting.
- Latency Middleware: Network latency was the cause of the small delays that appeared in the Heroku queue. Most of the time, the delay was reduced to about 10 seconds by the asynchronous call of the functions.
- Data Synchronization: The problem of AI cache that results in different data from those displayed in Lightning has been solved by using Lightning Data Service for event-driven synchronization.

Essentially, the hybrid integration was a potent, versatile, and user-centric architecture that seamlessly prolonged the AI callouts while still complying with Salesforce's performance and security standards.

5. Results and Discussion

5.1. Key Findings

The gradual shift to the hybrid Visualforce–Lightning architecture has reflected spectacularly in various areas such as API callout efficiency, stability, and user experience. As a matter of fact, the quantitative experimental results which have been thoroughly discussed in the previous section, have substantiated the new model’s capability of cutting the average response time by about 32%, of reducing the failure of the timeout by more than 80%, and of increasing to more than 90% of the overall API success rates. So, these facts are a clear indication that the addition of the Visualforce layer as a kind of mediator considerably alleviates the problem of the very limited nature of Salesforce’s Lightning execution environment.

It was through more prolonged server-side processing and less client session dependency that the Visualforce-mediated callouts managed to bring about these changes. In contrast, Lightning Components are affected by browser-level timeouts and have short transaction windows. Hence, the hybrid model that involves Visualforce managed to get around these restrictions by going straight for the server, thus, it can be said that the callout execution was delegating to the server via Visualforce. As a result, the AI-driven tasks such as lead scoring or sentiment analysis are not interrupted anymore. Besides that, the server-side operations also cut down on the number of requests that fail due to network delays or the browser not performing well.

On the scale of scalability, the hybrid solution was capable of being effective across various loads of work and adaptable. The Visualforce controller in high-traffic scenarios could execute AI requests that are asynchronous using the Apex processes, thus, operations that are concurrent can be done without the system resources being overwhelmed. The system was linearly scalable for moderate transaction volumes, which means that performance degradation was minimal even when the number of simultaneous API requests was high. This finding is very important for large-scale Salesforce deployments where multiple users can concurrently initiate AI interactions.

In the case of the hybrid layout, the topic of maintainability modular programming is supported. While the user interface is still managed by Lightning Components, Visualforce pages take over server communication and business logic. This segregation of work makes it simpler for the developers to find the bug and upgrade, since they are able to change the backend logic without having to alter the front-end. Besides that, the integration conformed to Salesforce's governor limits as it guaranteed that each callout was within the allowed thresholds even if they were distributed over different Visualforce sessions.

Another point to resource utilization and response stability. The server logs showed that the hybrid model had more stable CPU and memory usage than the Lightning-only model. The Visualforce page was like a controlled execution environment that lowered the overhead caused by the repeated re-rendering of client components. API response stability got significantly better: latency variability (the change between the fastest and slowest responses) was reduced by almost 40% thus, AI output delivery became more predictable and reliable. These changes emphasize the hybrid model’s success in sustaining platform performance while granting advanced AI capabilities.

5.2. Comparative Analysis

A comprehensive comparative evaluation was made between the proposed hybrid architecture and conventional Salesforce integration methods. These included standard Apex callouts and asynchronous Apex handling (Future, Queueable, and Batch Apex). In fact, the hybrid approach was the winner

of most of the performance and reliability dimensions measured, according to the outcomes.

Traditional Apex callouts suffer from the limitation of their synchronous execution window. On the one hand, when a Lightning Component fires off a callout directly, it has to wait for the result before moving on, hence, the interface is blocked and during long processing user experience deteriorates. On the other hand, asynchronous solutions like Queueable Apex can alleviate the problem but, at the same time, they introduce a delay and do not support real-time feedback. For its part, the Visualforce-mediated method leverages asynchronous execution and at the same time allows continuous interaction. Indeed, users could trigger a callout, get status updates via event-driven notifications, and see the results instantly, all without having to leave the Lightning interface.

Under the same conditions, the Visualforce-hybrid solution was a far cry from traditional callouts with regard to the performance metrics. The success rates went up from 68% to 93% and the number of transaction retries was cut down by a large margin. Furthermore, the hybrid pattern behaved like a time bomb under different network latency conditions, i.e., it was able to maintain the service continuity whereas the standard callouts were losing their connection most of the time. What is more, they also retained Salesforce's rigorous execution compliance criteria, thus they did not break any governor limits even when they were load tested.

The accomplishments, however, only partially. There was a need for additional work due to Visualforce middleware component which was added to the architecture, session management, cross-domain communication, and security compliance (especially with Content Security Policy restrictions) had to be configured. The developers had to work with two layers - the Lightning front end and the Visualforce backend - at the same time and, therefore, they had to be experts in both frameworks. On top of that, the necessity for asynchronous Apex jobs and caching processes implied that there were some additional overheads that needed to be taken into account for the entire implementation.

In spite of the intricacies, the improvements in performance and the stability enhancements are the reasons why the trade-off is acceptable in most enterprise scenarios. The hybrid architecture is transforming Salesforce from a reactive, time-bound system into a platform that can keep sustained AI interaction, which is mandatory for CRM operations of the future relying on predictive analytics, natural language processing, and generative AI.

What is more, the user point of view is on a par with the importance of such change. The hybrid method made the Lightning interface responsive again by separating heavy AI computations from the UI thread. Users spoke about more fluid interactions, shorter feedback loops, and lesser disruptions. System administrators witnessed a better monitoring and maintenance process due to the logs and error rates showing improved consistency instead of fragmented retries as in standard implementations.

5.3. Practical Implications

The successful validation of the hybrid architecture, which is a big move, comes with a bunch of real-life implications for Salesforce developers, administrators, and organizations who are thinking about how to use AI-driven capabilities to make their CRM ecosystem more powerful.

Firstly, the method improves user experience in a very elegant way. The application responsiveness in real-time which is a must for Lightning applications is preserved even though extended-duration AI workflows are supported. Users are not stuck with interfaces that do not react or loading states that are not understandable. Rather, they get clear status updates and predictable finishing times. This development means a lot, for instance, in sales and service environments where the level of efficiency and speed of feedback are the things that directly influence user satisfaction and productivity.

Secondly, the suggested design offers developers more possibilities. By using Visualforce as a layer that is not only for the front end but can be controlled for processing, developers are granted the liberty of going beyond the limitations of Salesforce’s synchronous callout limits to implement a wider variety of AI or data-intensive services. On the other hand, the same strategy can be utilized in other fields requiring heavy computation or communication with an external system, for example, document classification, image recognition, or natural language summarization.

Besides AI, the hybrid model paves the way for a large number of potential cross-platform integrations. To give a few examples, ERP systems, such as SAP or Oracle, that usually require long data synchronization sessions, can find the same architecture very useful. In the same way, Internet of Things (IoT) solutions the scenario in which device data may be streamed at intervals or has to be aggregated are able to employ Visualforce as a kind of intermediating agent to manage longer communication sessions with external gateways. Such adaptability makes it possible for the hybrid model to serve as a common design pattern that can be reused across different Salesforce integration projects.

In terms of operations, the architecture is a great example of how the company can scale up over a period of time. Aftenforce is constantly making changes to its API and execution engines, the modular delineation between Lightning (UI) and Visualforce (server processing) is such that changes made to one layer hardly affect the other. This modularity empowers the company to keep on innovating without risking their current setups or the experiences of the users.

The hybrid approach, in fact, is in line with Salesforce's strategic move towards AI-augmented CRM experiences. When organizations decorate their daily workflows with generative and predictive intelligence, the capability to maintain stable, long-running AI interactions will be a must. The Visualforce-Lighting hybrid is a real, law-abiding solution to this problem while at the same time it is in accordance with Salesforce's fundamental architectural principles of multi-tenancy, security, and performance governance.

6. Conclusion and Future Scope

The limitation of API callout duration in Salesforce Lightning as the integration of advanced artificial intelligence (AI) or machine learning (ML) systems was a challenge persisting for long, is what this study tried to solve. Through the creation and assessment of a hybrid architecture embedding Visualforce pages within Lightning Components, the paper essentially conveys extending the AI callout execution time without a breach of Salesforce's governor limits by this mix. As a result of a detailed case study, structured around an AI-based lead scoring system, the suggested model made significant improvements in performance: lower average response times, greater execution rates, and considerably fewer timeout errors. The method also confirmed that using Visualforce as a server-side mediator ensures processing continuity and stability, at the same time preserving Lightning's responsiveness and modern user experience.

The survey results reveal that Visualforce continues to be an integral part of the contemporary Salesforce ecosystem and hence is still relevant despite the rise of Lightning Web Components (LWC) and client-driven architectures. While Lightning offers a refined, modular front-end experience, Visualforce is still indispensable for dealing with complex or extensive backend operations. The study here is that Visualforce is not an obsolete technology that has been phased out by the new Lightning but rather it is a continuation of Salesforce's client interface that can interact with external AI systems. As Salesforce continues to enhance its platform with features like Dynamic Interactions, Lightning Data Service, and improved API concurrency, these hybrid approaches will be the way to achieve a performance, scalability, and compliance balance in enterprise environments.

This piece of work can be indefinitely extended in the future in several different ways. Later versions could rely on Salesforce Functions or an event-driven architecture to separate the processing from the user interface even further, thus allowing AI interactions to be almost in real-time without the server load limitation. The use of serverless AI connectors through something like AWS Lambda or MuleSoft, which can be powered by a platform like Salesforce, would allow for great scalability and less infrastructure overhead. Also, the alignment of this architecture with Salesforce's low-code/no-code model will empower administrators to set up AI features via declarative tools, thus, developers will not need to be authorized. In short, the deeper integration with Einstein AI and Salesforce Data Cloud can reveal more profound predictive insights, thus, the hybrid model can become a dominant pattern for the next generation of intelligent, self-learning CRM ecosystems.

References

1. Srinivas, Noone, Nagaraj Mandaloju, and Siddhartha Varma Nadimpalli. "Overcoming Challenges in Salesforce Lightning Testing with AI Solutions." Journal of Advanced Computing Systems 4.4 (2024): 1-12.
2. Guduru, Venkat Sumanth. "MASTERING SALESFORCE CLASSIC TO LIGHTNING MIGRATION: A COMPLETE GUIDE."
3. Yin, Junjie. "Salesforce-Usability of Lightning Web Components." (2019).
4. Cheng, Anson. Lightning prediction for space launch using machine learning based off of electric field mills and lightning detection and ranging data. No. AFITENSMS20M138. 2020.
5. Weinmeister, Philip. Practical Guide to Salesforce Communities. Apress:, 2018.
6. Nowacki, Krzysztof. Salesforce CRM Administration Handbook. 2024.
7. Carlos, Martínez, and Gómez Sofía. "AI-Powered CRM Solutions: Salesforce's Data Cloud as a Blueprint for Future Customer Interactions." International Journal of Trend in Scientific Research and Development 6.6 (2022): 2331-2346.
8. Nur, Renza. "AI-Powered Code Quality and Vulnerability Analysis in Salesforce DX: A DevSecOps Approach to Secure Application Lifecycle Management." (2024).
9. Sardjono, Wahyu, Achmad Cholidin, and Johan Johan. "Implementation of artificial intelligence-based customer relationship management for telecommunication companies." E3S Web of Conferences. Vol. 388. EDP Sciences, 2023.
10. Pentyala, Dillep Kumar. "Artificial intelligence for fault detection in cloud-optimized data engineering systems." International Journal of Social Trends 2.4 (2024): 8-44.
11. Parmar, Dipal. "Enhancing customer relationship management with salesforce Einstein GPT." (2023).
12. Garg, Shivansh, and Himanshu Jindal. "Operation Theater Allocation System using Salesforce CRM." (2023).
13. Ezzat, Rania. "Enhance the advertising effectiveness by using artificial intelligence (AI)." Journal of Art, Design and Music 3.1 (2024): 1.
14. Khanine, Dmitri. Optimizing Salesforce Industries Solutions on the Vlocity OmniStudio Platform: Implementing OmniStudio best practices for achieving maximum performance. Packt Publishing Ltd, 2024.
15. Shaalan, Sharif. Salesforce for Beginners: A step-by-step guide to creating, managing, and automating sales and marketing processes. Packt Publishing Ltd, 2020.