Study on the Implementation of AI Inference Services in a Business Environment

Nicolás Pérez-Leis, David Lacalle, and Paula M. Castro

Technology Department, Inditex, 15143 Arteixo, Spain
Technology Department, Inditex, 15143 Arteixo, Spain
Electronic Technology and Communications Group & CITIC Research Center, Universidade da Coruña, 15008 A Coruña, Spain
Correspondence: paula.castro@udc.es

DOI: *https://doi.org/10.17979/spu.23.c33*

Abstract: Diffusion-based generative models, such as Stable Diffusion, have revolutionized the generation of images from textual descriptions. However, their high computational cost represents an obstacle in production environments. This work analyzes strategies for their efficient and reproducible deployment in scalable clusters, with the technical support and resources provided by the company. The study is conducted in a real corporate context, where these models are already being used to accelerate design processes and optimize key stages of the value chain. To achieve this, different inference service strategies are compared in terms of latency, GPU utilization, and integration capabilities.

1 Introduction

In recent years, artificial intelligence has gone from being an analysis tool to becoming a direct source of content. Automatic generation of text, video, and images has transcended research laboratories to become integrated into creative processes, sales platforms, and decision-making. Among the areas where this transition is most evident is the generation of images from text, a field in which diffusion models (Yang et al., 2022) have had a notable impact due to their open, flexible, and efficient nature.

These models are increasingly used to support design or marketing teams in exploring visual ideas, creating promotional content, or prototyping graphic materials without the need for traditional photographic resources. In sectors such as the textile industry, the ability to generate relevant images quickly and on demand is a strategic advantage for shortening creative cycles, customizing products, and enriching the digital shopping experience. However, their practical adoption presents significant challenges: deploying a model such as Stable Diffusion (SD) (Wikipedia contributors, 2025) in production requires GPU resources, low latency, and a scalable infrastructure capable of adapting to load variations without affecting performance or increasing costs.

The choice of service architecture is the key to avoiding operational bottlenecks. There are multiple options for implementing inference, from lightweight frameworks such as FastAPI to advanced solutions such as Ray Serve or Lit Serve, focused on scaling and distributed management. However, the lack of comparative references on their effectiveness under real loads makes decision-making difficult in corporate environments.

Beyond its technical dimension, this work adds value by analyzing how to efficiently deploy generative AI models in real-world contexts, reducing response times and computational consumption. This optimization not only facilitates the integration of these technologies in any

company, but also contributes to technological sustainability, aligning with global goals such as the Sustainable Development Goals (SDGs) through more responsible energy use.

The article is structured as follows: Section 2 describes SD; Section 3 introduces the main technologies used, and Section 4 explains the development methodology; Section 5 shows the test results, and finally, Section 6 briefly presents the conclusions of the work.

2 Stable Diffusion

In recent years, diffusion models have established themselves as one of the most effective approaches in generative artificial intelligence. Their operation is based on a two-phase probabilistic process: progressive degradation, where noise is added to a real image, and inverse reconstruction, in which a neural network attempts to recover the original image from the noise. A key optimization is Latent Diffusion Models (LDMs) (Rombach et al., 2021), which apply diffusion in a latent space rather than directly on pixels, significantly reducing computational cost without compromising visual quality, which favors their use in production environments with limited resources. Stable Diffusion is an open-source text-to-image model released on August 22, 2022, by a consortium comprising CompVis and Runway AI, with the backing of Stability AI. The model uses the LDM family, where the process is carried out in a compressed latent space to reduce resources and speed up inference. The architecture is divided into three main blocks (see Figure 1):

- VAEs: encode images into latent space and decode them at the end of the process (EITCA Academy, 2024).
- U-Net network: progressively removes noise from the latent representation (Ronneberger et al., 2015).
- CLIP ViT-L/14 text encoder: transforms the prompt into an embedding that guides the U-Net through cross-attention (Komorebi AI, 2022).

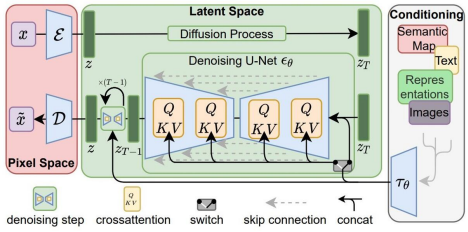

Figure 1: SD architecture.

SD v1.4 is the usual starting point for users and projects, due to its aesthetic consistency in artistic prompts, robustness against overfitting, and compatibility with extensions such as LoRA or DreamBooth.

3 Tools and Technologies

The technologies used enable the deployment of artificial intelligence services accessible via API, creating scalable environments and facilitating access to data by other departments.

3.1 Docker

Docker (Google Cloud, 2025) is a platform that packages applications and dependencies into containers to ensure a consistent environment. It has been used to isolate the deployment of

SD and other services, improving reproducibility and dependency management. The process includes creating a Dockerfile with the necessary actions and dependencies, followed by obtaining the image for the final deployment.

3.2 Kubernetes

Kubernetes (Kubernetes contributors, 2025) is a container orchestration system that automates the management, deployment, and scaling of applications in cluster environments. In this work, Kubernetes has been used, through Rancher Desktop, to implement a local cluster to perform load and performance tests on the SD pipeline in distributed environments, ensuring the scalability of the service. In Kubernetes, the cluster is structured into nodes, which in turn contain pods, which can host one or more containers where the services are executed.

3.3 Terraform

Infrastructure as code (IaC) is a common practice in data engineering and artificial intelligence. In this field, Terraform (HashiCorp, 2025) is used to define and deploy environments in an automated, reproducible, and versioned manner.

By combining Terraform with Google Kubernetes Engine (GKE), it is possible to declaratively configure the size and type of nodes, the network, storage, access policies, as well as project services and permissions. This facilitates the creation of isolated environments for testing and allows for the automated deployment of production environments from continuous integration and delivery (CI/CD) pipelines.

3.4 FastAPI

FastAPIs (FastAPI contributors, 2025) is a Python library for creating high-performance APIs that are easy to develop and easy to scale. It has been used to develop the SD pipeline, demonstrating its efficiency in handling concurrent requests. This tool provides automatic documentation, which facilitates the use of the model by third parties.

3.5 Litserve

Litserve (Lightning AI contributors, 2025) is a Lightning AI library that facilitates the creation of servers to deploy machine learning models. During the project, it was used to configure a server that returned images generated by the SD model (similar to that done with FastAPI), allowing easy access to the model without users needing to know its inner workings.

3.6 Rayserve

Rayserve (Ray Project contributors, 2025) is a distributed services platform for deploying and inferencing large-scale machine learning models. Rayserve is explored to test the deployment of the SD model on clusters with more powerful hardware, comparing its performance with other tools, such as Litserve and FastAPI.

3.7 Locust

Locust (Locust contributors, 2025) is a load tool that simulates user traffic to test the performance of the application. It is used to perform load tests on the SD server, measuring response times and efficiency under different conditions of use, with the aim of optimizing service performance. Locust provides a graphical interface to see how the service behaves when faced with external requests.

4 Method

For this project, a waterfall methodology was adopted, structured in six sequential phases (see Table 1), but with internal iterations to review technical decisions or resolve blockages.

Table 1: Approximate time distribution of the project phases

Project phase	Duration (days; 4 hours/day)
Phase 1 – Preliminary research and local testing	25
Phase 2 – Infrastructure design and provision	20
Phase 3 – Inference service development	35
Phase 4 – Design and execution of load testing	15
Phase 5 – Drafting of results and technical documentation	10
Phase 6 – Development of visual interface for inference	10

The comparison between frameworks was based on key metrics extracted from Locust tests. The number of requests per second (RPS) was primarily evaluated as a direct indicator of performance and concurrent processing capacity. At the same time, latency was monitored, emphasizing the 50th and 95th percentiles to reflect both average behavior and cases under higher load. Errors per second were also recorded to detect possible bottlenecks or instabilities in high-concurrency situations. These metrics, obtained under controlled and homogeneous conditions, formed the basis for a comparative evaluation of the performance of each inference solution. In addition, qualitative aspects such as ease of integration, flexibility, and suitability of each approach to various technical scenarios were considered.

The metrics presented are based on more than 3,000 inferences per framework, generated through automated concurrent requests from Locust. This volume allowed us to identify performance patterns and potential bottlenecks under realistic conditions.

Each request contained a random prompt to prevent the reuse of results and ensure complete execution of the pipeline without computational shortcuts. The requests, sent in JSON format to the /predict endpoint, automatically discarded incomplete or erroneous inferences. All were performed with four inference steps, focusing the analysis on quantitative response and performance metrics, not on the quality of the generated images.

The data used was exclusively technical, without including personal or sensitive information, using synthetic data generated in real time. The resulting images were not stored and the analysis focused on the behavior of the system under different concurrent loads, ensuring a controlled and reproducible scenario. This methodology provided a considerable volume of reliable observations to support the analysis of results.

5 Results

This section analyzes the results of the three solutions mentioned in Section 3 for serving inferences from the SD v1.4 model: FastAPI, Lit Serve, and Ray Serve. Their performance, stability, and viability in real-world environments were compared using load testing with concurrent requests.

The tests were performed on a uniform infrastructure with three replicas per service, each running on an NVIDIA L4 GPU, ensuring comparable conditions. The models were converted to float16 precision, which reduced inference times by approximately 50 % without affecting quality.

5.1 *FastAPI* Evaluation

Figure 2 shows the results obtained during a staggered load test with Locust, in which the number of concurrent users was progressively increased to 30.

The service reached a stable average of around 0.9 RPS with peaks of up to 1.3 RPS at certain intervals. In terms of latency, consistent times of around 12–15 seconds were observed at the 50th percentile and up to 25–30 seconds at the 95th percentile, as the load increased. Although the response remains reasonable for a model of this type, some isolated failures were detected that could be related to GPU saturation or timeouts in the management of simultaneous requests.

FastAPI's overall performance was stable, with controlled degradation under stress and no critical interruptions, making it a valid option for low-volume environments or scenarios where ease of integration is prioritised over scalability or internal visibility of the model.

5.2 *Lit Serve* Evaluation

Figure 3 shows the system's behaviour during a load test with up to 100 virtual users and staggered requests.

Compared to *FastAPI*, *Lit Serve* offers a considerable improvement in performance: sustained rates of up to 6 RPS are achieved with no errors recorded. This efficiency is also reflected in response times, with values in the 50th percentile close to 2,500 ms, and in the 95th percentile around 8,000 ms. These figures are lower than those observed in FastAPI, even with a higher user load, highlighting better internal concurrency management.

Therefore, despite its limitations in defining multiple endpoints, Lit Serve is competitive in environments where efficiency and a clear functional structure are required to encapsulate the model logic.

5.3 *Ray Serve* Evaluation

Figure 4 shows the performance observed during a progressive load test with up to 24 concurrent users.

During the test, the system reached a stable rate of approximately 4.3 RPS, with 0 failures recorded. This value, although lower than that achieved with Lit Serve, is consistent with a more controlled and distributed approach to load balancing among internal actors. The response times remained relatively low and stable. At the 50th percentile, latencies were around 4,500 ms, while at the 95th percentile they did not exceed 7,300 ms. This contained difference between percentiles suggests predictable behaviour and good concurrency management.

It should be noted that, compared to FastAPI, Ray Serve showed significantly lower and more consistent latency, as its internal architecture allows for more efficient distribution of inference processes among active replicas. Overall, the results of Ray Serve demonstrate a robust and stable solution, particularly suitable for environments where reliability and load distribution across multiple GPUs is a key requirement.

6 Conclusions

The results not only compare the performance of each inference solution, but also reflect the technical rigour applied to ensure a reproducible and representative evaluation. The complete automation of the testing environment made it possible to isolate the behaviour of each framework under controlled conditions, ensuring objective comparability.

This study constitutes a systematic validation in a realistic environment, managed with Terraform and Kubernetes on dedicated GPUs, establishing a solid foundation for efficiently integrating generative image models into production.

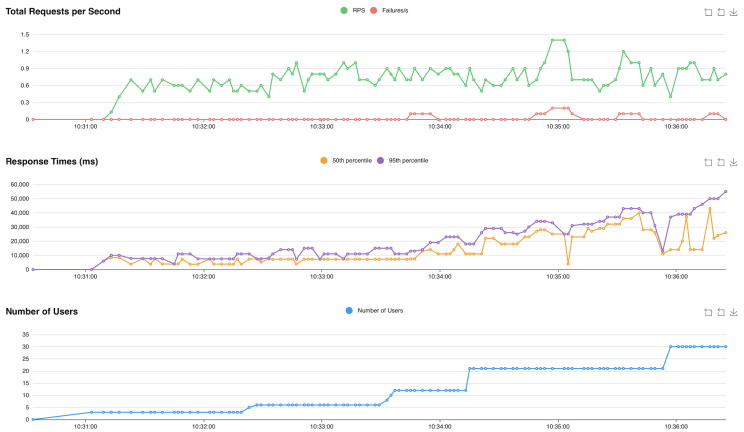

Figure 2: FastAPI

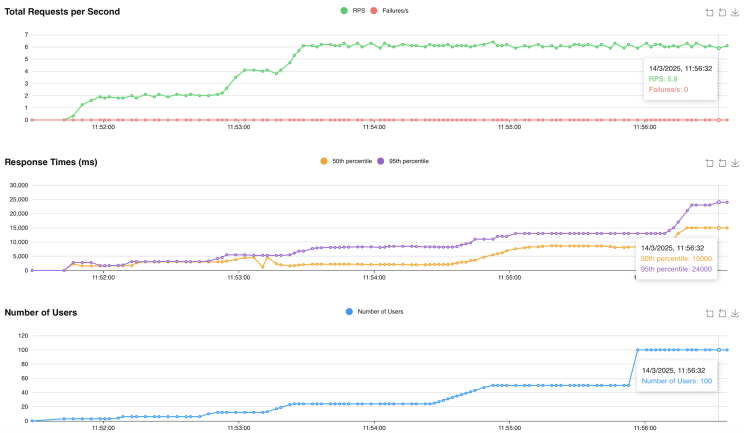

Figure 3: Lit Serve

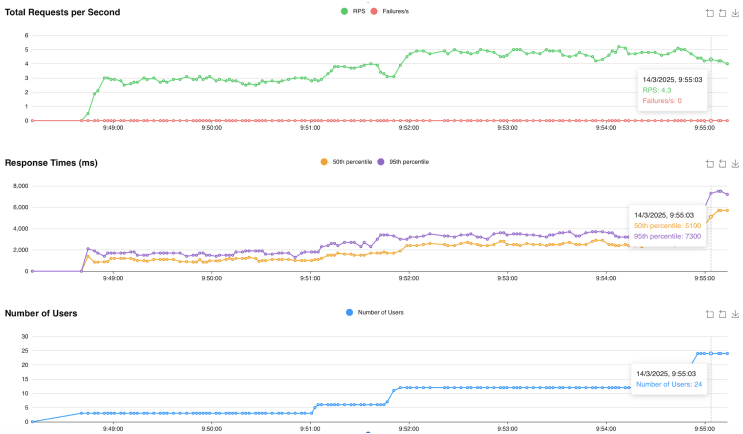

Figure 4: Ray Serve

Acknowledgements

This work has been funded with the support from ED431C 2024/18 of Xunta de Galicia.

Bibliography

EITCA Academy. What is the reparameterization trick and why is it crucial for the training of variational autoencoders (vaes)? EITCA Academy – Advanced Generative Models, 2024. [En línea]. Disponible en: *https://es.eitca.org/artificial-intelligence/eitc-ai-adl-advanced-deep-learning/.../what-is-the-reparameterization-trick-and-why-is-it-crucial-for-the-training-of-variational-autoencoders-vaes/*.

FastAPI contributors. Fastapi documentation. FastAPI, Tiangolo, 2025. [En línea]. Disponible en: *https://fastapi.tiangolo.com*.

Google Cloud. Store docker container images in artifact registry. Google Cloud Artifact Registry documentation, 2025. [En línea]. Disponible en: *https://cloud.google.com/artifact-registry/docs/docker/store-docker-container-images*.

HashiCorp. Get started - terraform for google cloud platform. HashiCorp Developer Tutorials, 2025. [En línea]. Disponible en: *https://developer.hashicorp.com/terraform/tutorials/gcp-get-started*.

Komorebi AI. Una introducción al aprendizaje contrastivo con clip. Komorebi AI Blog, 2022. [En línea]. Disponible en: *https://komorebi.ai/es/una-introduccion-al-aprendizaje-contrastivo-con-clip/*.

Kubernetes contributors. Learn kubernetes basics. Kubernetes Documentation Tutorial, 2025. [En línea]. Disponible en: *https://kubernetes.io/docs/tutorials/kubernetes-basics/*.

Lightning AI contributors. Litserve features. Lightning AI Docs, 2025. [En línea]. Disponible en: *https://lightning.ai/docs/litserve/features*.

Locust contributors. What is locust? Locust Documentation, 2025. [En línea]. Disponible en: *https://docs.locust.io/en/stable/what-is-locust.html*.

Ray Project contributors. Ray serve documentation. Ray Docs, 2025. [En línea]. Disponible en: *https://docs.ray.io/en/latest/serve/index.html*.

R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. arXiv preprint arXiv:2112.10752, 2021. [En línea]. Disponible en: *https://arxiv.org/abs/2112.10752*.

O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. arXiv preprint arXiv:1505.04597, 2015. [En línea]. Disponible en: *https://arxiv.org/abs/1505.04597*.

Wikipedia contributors. Stable diffusion. Wikipedia, The Free Encyclopedia, 2025. [En línea]. Disponible en: *https://en.wikipedia.org/wiki/Stable_Diffusion*.

L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui, and M.-H. Yang. Diffusion models: A comprehensive survey of methods and applications. arXiv preprint arXiv:2209.00796, 2022. [En línea]. Disponible en: *https://arxiv.org/abs/2209.00796*.