UNIVERSIDAD CATÓLICA DE CUENCA

UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

FACULTAD DE INFORMÁTICA, CIENCIAS DE LA COMPUTACIÓN E INNOVACIÓN TECNOLÓGICA

CARRERA DE SOFTWARE

IMPLEMENTACIÓN DE UN MICROSERVICIO ESCALABLE PARA EL ANÁLISIS AUTOMATIZADO DE CÁNCER CEREBRAL EN IMÁGENES DE RESONANCIA MAGNÉTICA MEDIANTE ARQUITECTURA DE CONTENEDORES.

PROYECYO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TITULO DE INGENIERO DE SOFTWARE

AUTORES: MATEO JOSUÉ BRAVO FERNÁNDEZ

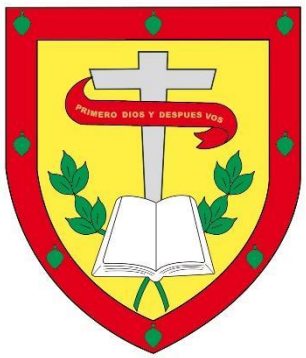
CARLOS ANDRÉS ALVARADO GUTIÉRREZ

DIRECTOR: ING. JOSÉ ANTONIO BACULIMA SUÁREZ, MGS.

CUENCA- ECUADOR

2026

DIOS, PATRIA, CULTURA Y DESARROLLO



UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

FACULTAD DE INFORMÁTICA, CIENCIAS DE LA COMPUTACIÓN E INNOVACIÓN TECNOLÓGICA

CARRERA DE SOFTWARE

IMPLEMENTACIÓN DE UN MICROSERVICIO ESCALABLE PARA EL ANÁLISIS AUTOMATIZADO DE CÁNCER CEREBRAL EN IMÁGENES DE RESONANCIA MAGNÉTICA MEDIANTE ARQUITECTURA DE CONTENEDORES.

PROYECYO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TITULO DE INGENIERO DE SOFTWARE

AUTORES: MATEO JOSUÉ BRAVO FERNÁNDEZ

CARLOS ANDRÉS ALVARADO GUTIÉRREZ

DIRECTOR: ING. JOSÉ ANTONIO BACULIMA SUÁREZ, MGS.

CUENCA- ECUADOR

2026

DIOS, PATRIA, CULTURA Y DESARROLLO

Universidad Católica de Cuenca	**DECLARATORIA DE AUTORÍA Y RESPONSABILIDAD**

Mateo Josué Bravo Fernández portador de la cédula de ciudadanía Nº **0106758477**. Declaro ser el autor de la obra: **"IMPLEMENTACIÓN DE UN MICROSERVICIO ESCALABLE PARA EL ANÁLISIS AUTOMATIZADO DE CÁNCER CEREBRAL EN IMÁGENES DE RESONANCIA MAGNÉTICA MEDIANTE ARQUITECTURA DE CONTENEDORES"**, sobre la cual me hago responsable sobre las opiniones, versiones e ideas expresadas. Declaro que la misma ha sido elaborada respetando los derechos de propiedad intelectual de terceros y eximo a la Universidad Católica de Cuenca sobre cualquier reclamación que pudiera existir al respecto. Declaro finalmente que mi obra ha sido realizada cumpliendo con todos los requisitos legales, éticos y bioéticos de investigación, que la misma no incumple con la normativa nacional e internacional en el área específica de investigación, sobre la que también me responsabilizo y eximo a la Universidad Católica de Cuenca de toda reclamación al respecto.

Cuenca, **08 de abril de 2026**

F: ……..

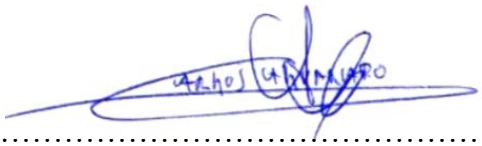
Mateo Josué Bravo Fernández

C.I. **0106758477**

Universidad Católica de Cuenca	**DECLARATORIA DE AUTORÍA Y RESPONSABILIDAD**

Carlos Andrés Alvarado Gutiérrez portador de la cédula de ciudadanía Nº **0104134762**. Declaro ser el autor de la obra: **"IMPLEMENTACIÓN DE UN MICROSERVICIO ESCALABLE PARA EL ANÁLISIS AUTOMATIZADO DE CÁNCER CEREBRAL EN IMÁGENES DE RESONANCIA MAGNÉTICA MEDIANTE ARQUITECTURA DE CONTENEDORES"**, sobre la cual me hago responsable sobre las opiniones, versiones e ideas expresadas. Declaro que la misma ha sido elaborada respetando los derechos de propiedad intelectual de terceros y eximo a la Universidad Católica de Cuenca sobre cualquier reclamación que pudiera existir al respecto. Declaro finalmente que mi obra ha sido realizada cumpliendo con todos los requisitos legales, éticos y bioéticos de investigación, que la misma no incumple con la normativa nacional e internacional en el área específica de investigación, sobre la que también me responsabilizo y eximo a la Universidad Católica de Cuenca de toda reclamación al respecto.

Cuenca, **08 de abril de 2026**



F: ……..

Carlos Andrés Alvarado Gutiérrez

C.I. **0104134762**

Universidad Católica de Cuenca	**CERTIFICADO DEL TUTOR**

Yo, José Antonio Baculima Suárez, certifico que el presente trabajo de investigación, con el título "**IMPLEMENTACIÓN DE UN MICROSERVICIO ESCALABLE PARA EL ANÁLISIS AUTOMATIZADO DE CÁNCER CEREBRAL EN IMÁGENES DE RESONANCIA MAGNÉTICA MEDIANTE ARQUITECTURA DE CONTENEDORES**", fue desarrollado por Mateo Josué Bravo Fernández con número de cédula 0106758477 y Carlos Andrés Alvarado Gutiérrez con número de cédula 0104134762, bajo mi supervisión.

Firmado electrónicamente por:
JOSE ANTONIO BACULIMA SUAREZ
Validar únicamente con FirmaEC

F: ……..

José Antonio Baculima Suárez

C.I. **0103638664**

Dedicatoria

Dedico este logro a mis padres, porque han sido un pilar muy importante para mi, han sido mi guía, mi fortaleza y mi inspiración a lo largo de toda mi vida. Con su apoyo incondicional me han enseñado el valor del esfuerzo, la dedicación, y que con su ejemplo me mostraron que los sueños se alcanzan paso a paso. Gracias por cada consejo, cada palabra de aliento y cada sacrificio que han hecho posible que hoy pueda alcanzar esta meta. Gracias por creer en mí cuando yo dudaba, por ayudarme en los momentos difíciles y por enseñarme que la constancia y la disciplina son las bases de todo logro verdadero.

A mi familia, por su acompañamiento paciente y por celebrar mis avances y no dejar desanimarme en los tropiezos. Este triunfo lleva consigo su cariño y su ejemplo, porque cada momento de dedicación y cada uno de mis logros han sido posibles gracias a la fe que me han brindado siempre.

Mateo Josué Bravo Fernández.

Dedico esta investigación a mi madre y a mi familia, quienes han sido guía, fortaleza e inspiración permanente en mi vida.

Este logro también les pertenece, porque cada paso dado estuvo acompañado de su confianza y amor.

Carlos Andrés Alvarado G.

Agradecimientos

Agradezco a Dios por darme la fortaleza y la sabiduría para culminar esta etapa de mi formación profesional.

A mi madre, por su amor, apoyo incondicional y por ser el pilar fundamental en cada paso de mi vida. Su confianza y ejemplo fueron esenciales para alcanzar esta meta.

Carlos Andrés Alvarado G.

Comenzaré dando gracias a aquellas personas que me acompañaron en cada paso de esta etapa. En primer lugar, quiero agradecer a mis padres por su apoyo sin límites, por su comprensión y por confiar en mí. Agradezco por estar en mis momentos difíciles y por cada palabra de aliento. Este también es su logro, porque cada esfuerzo que hice estuvo impulsado por ustedes. También quiero dar gracias a mis hermanos, por estar a mi lado y darme fuerza cuando más los necesitaba. Su compañía ha sido un pilar importante en este camino. Doy gracias a mis amigos que me ofrecieron su ayuda, compartieron su tiempo y me acompañaron durante toda la carrera. A mis profesores de carrera y tutores de tesis, gracias por todos sus consejos y sus enseñanzas, han sido muy importantes para superar los desafíos y avanzar con confianza, agradezco por compartir sus conocimientos, motivándome a aprender y a crecer académicamente. A todos ustedes, les agradezco, este logro es el reflejo de la constancia, la guía y el acompañamiento que recibí a lo largo de mi formación académica.

Mateo Josué Bravo Fernández.

Esta investigación se realizó como parte del trabajo del grupo de investigación "Grupo de Investigación Multidisciplinar para la Innovación" (GIMI) en el marco del proyecto titulado "Localización e Identificación de Patologías en Imágenes Médicas a Través de Sistemas de Diagnóstico Automatizado con Redes Neuronales" (PICTMS24-30), perteneciente al Laboratorio de Robótica, Automatización, Sistemas Inteligentes y Embebidos (RobLab) de la Universidad Católica de Cuenca.

Resumen

El presente trabajo propone e implementa una arquitectura de microservicios para la operacionalización de modelos preentrenados de segmentación de tumores cerebrales en imágenes médicas. Si bien numerosos estudios se centran en el desempeño algorítmico de modelos de aprendizaje profundo, existe una brecha entre su validación experimental y su integración en sistemas técnicamente desplegables. En este contexto, se desarrolló un microservicio capaz de integrar modelos de inteligencia artificial preentrenados para el área de la salud, permitiendo su ejecución en distintos entornos de hardware.

El estudio se enfocó en la evaluación del desempeño computacional del sistema, considerando métricas como latencia de inferencia, consumo de memoria y utilización de GPU bajo diferentes configuraciones. Los resultados evidencian una dependencia significativa de la aceleración por hardware en modelos que evalúan imágenes tridimensionales, así como variaciones en los tiempos de respuesta según el entorno de ejecución.

La propuesta contribuye a cerrar la brecha entre modelos experimentales y sistemas desplegables, validando la viabilidad técnica de su integración en una arquitectura modular, desacoplada y potencialmente escalable. No se abordó validación clínica directa, delimitando el alcance a la evaluación computacional y estructural del sistema.

Palabras clave: *microservicios, arquitectura de contenedores, despliegue de modelos de IA, desempeño computacional, MLOps.*

Abstract

This study proposes and implements a microservices architecture for deploying pre-trained brain tumor segmentation models in medical imaging. While numerous studies focus on the algorithmic performance of deep learning models, a gap remains between their experimental validation and their integration into technically deployable systems. In this context, a microservice was developed to integrate pre-trained artificial intelligence models into the healthcare sector, enabling their execution across different hardware environments.

The study focused on evaluating the system's computational performance, considering metrics such as inference latency, memory consumption, and GPU utilization under different configurations. The results demonstrate a significant reliance on hardware acceleration for models that process three-dimensional images, as well as variations in response times depending on the execution environment.

The proposal helps to bridge the gap between experimental models and deployable systems by validating the technical feasibility of integrating them into a modular, decoupled, and potentially scalable architecture. Direct clinical validation was not addressed, limiting the scope to the computational and structural evaluation of the system.

Keywords: *microservices, containerized architecture, AI model deployment, computational performance, MLOps.*

Índice

CAPÍTULO I. INTRODUCCIÓN

1.1. Planteamiento del problema

En los últimos años, el desarrollo de modelos de inteligencia artificial aplicados al análisis de imágenes médicas ha experimentado un crecimiento significativo. Sin embargo, su integración efectiva en entornos clínicos reales continúa siendo limitada. Diversos estudios evidencian la existencia de una brecha considerable entre los modelos desarrollados en contextos de investigación y su implementación operativa en sistemas hospitalarios, debido principalmente a dificultades relacionadas con el despliegue, la interoperabilidad y la arquitectura de software utilizada (Gupta et al., 2024).

En particular, muchos sistemas actuales de análisis de imágenes médicas no incorporan arquitecturas basadas en microservicios que permitan el procesamiento distribuido de imágenes en formato DICOM (Digital Imaging and Communications in Medicine) mediante la utilización de modelos de deep learning previamente entrenados. La ausencia de servicios independientes para tareas específicas como la segmentación visual, la clasificación y la detección tumoral, limita la escalabilidad, la mantenibilidad y la capacidad de integración de estos sistemas en entornos clínicos complejos (Huang & Liu, 2025).

Asimismo, la integración de modelos de inteligencia artificial en sistemas clínicos presenta desafíos asociados al cumplimiento de estándares de interoperabilidad, como DICOM, y a su interacción con infraestructuras hospitalarias existentes, tales como los sistemas PACS (Picture Archiving and Communication System). Esta situación dificulta la adopción de soluciones inteligentes capaces de generar resultados visuales interpretables y métricas cuantitativas confiables que apoyen el proceso diagnóstico. En este contexto, los profesionales del área médica requieren herramientas capaces de analizar imágenes médicas mediante la comparación con modelos previamente entrenados, generando segmentaciones visuales con colores que se puedan interpretar y facilitando la detección automatizada de tumores, y el cálculo de mediciones morfométricas relevantes (Theriault-Lauzier et al., 2024).

Otro desafío importante se relaciona con la actualización y mantenimiento de los modelos de inteligencia artificial. En muchos sistemas tradicionales, la actualización de modelos implica detener el servicio, lo que genera interrupciones en la continuidad operativa de

los entornos clínicos. Adicionalmente, una gestión ineficiente de los recursos computacionales, particularmente GPU y memoria, pueden incrementar los costos operativos y generar subutilización del hardware especializado, limitando la viabilidad económica de estas soluciones a gran escala (Brundage et al., 2022; Huang & Liu, 2025).

1.2. Justificación

Relevancia Científica:

Desarrollar microservicios con el fin de analizar imágenes médicas a nivel comparativo es un aporte pertinente en el área de la informática médica. Estudios recientes señalan que la transición desde prototipos experimentales hacia arquitecturas escalables y mantenibles es uno de los principales desafíos para la maduración de la IA médica (Gupta et al., 2024; Theriault-Lauzier et al., 2024).

El desarrollo de sistemas capaces de comparar imágenes DICOM con modelos previamente entrenados permite avanzar desde entornos experimentales hacia herramientas potencialmente utilizables en contextos clínicos reales, con su propuesta de tener una arquitectura por módulos, que sea escalable y reutilizable, aplicando ingeniería moderna y MLOps aplicado a entornos médicos (Huang & Liu, 2025).

Relevancia Tecnológica:

Adoptar la arquitectura de microservicios es relevante al ámbito tecnológico debido a que es una tendencia emergente en el despliegue de soluciones médicas con IA. Algunos de los frameworks recientes tienen la necesidad de separar servicios de inferencia, monitoreo y actualización con el fin de brindar robustez y escalabilidad. Microservicios que integren motores de inferencia como TensorFlow/Keras bajo arquitecturas cloud-native potencia el procesamiento distribuido de imágenes DICOM y la gestión eficiente de recursos computacionales, y así estar apegado a las mejores prácticas actuales con respecto a infraestructura médica con IA (Brundage et al., 2022; Huang & Liu, 2025).

Relevancia Práctica:

Implementar microservicios en instituciones en donde se necesite un análisis automatizado de imágenes médicas aporta una solución eficaz e inmediata. Generando segmentaciones interpretables con colores y la detección tumoral automatizada con modelos entrenados que pueden realizar comparación, ayudan a mejorar la eficiencia diagnóstica reduciendo la carga operativa.

El déficit de infraestructuras adecuadas de despliegue es uno de los principales factores que limita la adopción clínica de la IA, incluso cuando los modelos presentan alto rendimiento en entornos experimentales. Por tanto, una arquitectura orientada a microservicios facilita la integración progresiva de IA en flujos hospitalarios reales (Gupta et al., 2024; Theriault-Lauzier et al., 2024).

Relevancia Económica:

La implementación de microservicios puede generar impactos económicos relevantes mediante la automatización de procesos en donde se requiera intervenir de manera manual. Las arquitecturas distribuidas y orientadas a eventos brindan eficiencia en la gestión del procesamiento DICOM en entornos cloud, por ejemplo optimizar la aceleración por hardware (uso de GPU) y reducir valores asociados a infraestructura sobredimensionada. Separar servicios facilita la actualización independiente de modelos sin tener interrupciones del sistema, reduce riesgos operativos y costos derivados de indisponibilidad tecnológica (Brundage et al., 2022; Huang & Liu, 2025).

1.3. Objetivos

1.3.1. Objetivo general

Diseñar e implementar una arquitectura basada en microservicios para la inferencia de modelos de segmentación de tumores cerebrales en imágenes de resonancia magnética, evaluando su desempeño computacional mediante métricas de latencia, consumo de memoria y utilización de aceleración por hardware (GPU), así como su capacidad de integración con sistemas clínicos existentes.

1.3.2. Objetivos específicos

- Realizar una revisión bibliográfica de arquitecturas y metodologías utilizadas en el desarrollo de microservicios que se aplican para analizar imágenes médicas, permitiendo la identificación de buenas prácticas en el uso de modelos de deep learning y mecanismos de comunicación entre servicios distribuidos.

- Diseñar e implementar microservicios especializados para el procesamiento de imágenes médicas en formato NIfTI/DICOM, integrando modelos de aprendizaje profundo preentrenados mediante una arquitectura multicapa que separe las responsabilidades de presentación, negocio, inferencia y persistencia.

- Desarrollar una arquitectura de contenedorización y coordinación utilizando tecnologías de virtualización ligera para el despliegue distribuido del microservicio en distintos entornos de hardware, estableciendo protocolos de comunicación y mecanismos de seguridad que garanticen la integridad y confidencialidad de los datos durante el procesamiento.

- Evaluar el desempeño computacional del microservicio mediante métricas de latencia de inferencia, consumo de memoria, utilización de GPU y tasa de errores bajo distintas configuraciones de hardware y niveles de carga concurrente.

1.4. Preguntas de investigación

1.4.1 ¿Qué niveles de latencia de inferencia, consumo de memoria y utilización de GPU presenta una arquitectura basada en microservicios para la ejecución de modelos de aprendizaje profundo en imágenes médicas bajo distintas configuraciones de hardware?

1.4.2 ¿Cómo afecta la arquitectura de contenedorización al desempeño y estabilidad operativa del microservicio en entornos de desarrollo y producción con diferente capacidad de cómputo?

1.4.3 ¿Qué mecanismos de seguridad y validación de entradas permiten garantizar la confidencialidad, integridad y estabilidad operativa del microservicio durante el procesamiento de imágenes médicas?

1.5. Hipótesis de la investigación

La implementación de una arquitectura basada en microservicios para la inferencia de modelos de segmentación de tumores cerebrales permite alcanzar niveles adecuados de latencia, estabilidad operativa y gestión eficiente de recursos computacionales (CPU, GPU y memoria RAM) en distintos entornos de hardware, demostrando su viabilidad técnica como infraestructura de despliegue para sistemas de análisis automatizado de imágenes médicas.

CAPÍTULO II. REVISIÓN DE LA LITERATURA

2.1. Estado del arte

El desarrollo de sistemas de inteligencia artificial aplicados a la medicina ha impulsado la investigación en arquitecturas de software sean capaces de soportar procesos de inferencia complejos, altos volúmenes de datos médicos y requisitos estrictos de interoperabilidad. En este contexto, diversas investigaciones han explorado el uso de arquitecturas basadas en microservicios, infraestructuras cloud-native y mecanismos de observabilidad para facilitar el despliegue y la operación de modelos de aprendizaje profundo en entornos clínicos.

Plataformas de Microservicios para IA Médica:

La incorporación de modelos de inteligencia artificial en entornos clínicos reales ha motivado el desarrollo de arquitecturas de software basadas en microservicios y prácticas MLOps, con el objetivo de facilitar el despliegue, monitoreo y actualización de modelos en producción. Estudios han demostrado que la separación de servicios especializados, como inferencia, monitoreo y gestión de modelos, contribuye a mejorar la estabilidad, mantenibilidad y resiliencia de los sistemas médicos basados en inteligencia artificial (Faseeha et al., 2025; Panchal et al., 2023).

En este contexto, el framework MLXOps4Medic propone una arquitectura basada en microservicios que permite gestionar modelos de inteligencia artificial de manera independiente, integrando mecanismos de monitoreo y control de aplicaciones de análisis de imágenes médicas. Este enfoque demuestra la importancia de utilizar arquitecturas desacopladas para facilitar el mantenimiento y la evolución de sistemas clínicos distribuidos (Huang & Liu, 2025).

De manera similar, otras investigaciones relacionadas, han analizado el uso de arquitecturas modulares para el despliegue de modelos de segmentación y clasificación médica, evidenciando que separando responsabilidades de los componentes del sistema permite una mejora en la capacidad de escalamiento y reducir la complejidad operativa en entornos hospitalarios (Bathelt et al., 2025; Harshith et al., 2026).

Arquitecturas Cloud-Native para procesamiento de imágenes médicas:

Las arquitecturas cloud-native han adquirido un papel relevante en el desarrollo de aplicaciones médicas basadas en inteligencia artificial, debido a su capacidad para gestionar cargas de trabajo intensivas y procesar grandes volúmenes de datos clínicos de forma distribuida.

Investigaciones recientes han demostrado que el uso de infraestructuras orientadas a eventos facilita el procesamiento eficiente de imágenes médicas en formatos como DICOM dentro de entornos distribuidos.

En este contexto, (Brundage et al., 2022) proponen una arquitectura cloud-native orientada a eventos para la conversión e integración de datos DICOM, permitiendo realizar procesamiento distribuido y escalable en infraestructuras basadas en contenedores.

Este tipo de arquitecturas permite desacoplar los distintos componentes del sistema, favoreciendo la escalabilidad horizontal y facilitando la integración con servicios especializados para análisis de imágenes médicas y ejecución de modelos de inteligencia artificial.

Integración de inteligencia artificial en sistemas clínicos:

Uno de los principales desafíos en la implementación de soluciones de inteligencia artificial en salud consiste en la integración de los modelos dentro de infraestructuras clínicas existentes, como los servicios PACS (Picture Archiving and Communication System) y los repositorios de imágenes médicas basados en DICOM.

En este contexto, la plataforma PACS-AI propone una arquitectura que permite integrar modelos de inteligencia artificial directamente dentro de flujos clínicos reales, facilitando la interacción entre sistemas de almacenamiento de imágenes médicas y módulos de análisis automatizado. Este enfoque resalta la importancia de diseñar soluciones modulares capaces de interoperar con infraestructuras hospitalarias existentes (Theriault-Lauzier et al., 2024).

Estas investigaciones evidencian que la integración efectiva de inteligencia artificial en entornos clínicos requiere arquitecturas de software flexibles que permiten incorporar nuevos modelos sin alterar la infraestructura base de los sistemas hospitalarios.

Interoperabilidad y Estándares Médicos:

La interoperabilidad constituye un requisito fundamental para la adopción de sistemas de inteligencia artificial en entornos clínicos. Diversos estudios han analizado el uso de estándares médicos para facilitar la integración de servicios de análisis automatizado dentro de infraestructuras hospitalarias.

En particular, el estándar DICOM permite gestionar y transmitir imágenes médicas entre sistemas clínicos, mientras que el estándar FHIR (Fast Healthcare Interoperability Resources) facilita el intercambio de información clínica estructurada mediante interfaces basadas en APIs REST.

Investigaciones recientes han demostrado que la combinación de DICOMweb y FHIR permite integrar modelos de inteligencia artificial dentro de flujos clínicos interoperables, garantizando la compatibilidad con infraestructuras hospitalarias modernas y facilitando el acceso a datos médicos en entornos distribuidos (Tang et al., 2023).

Herramientas de Observabilidad Especializadas:

En arquitecturas basadas en microservicios, la observabilidad constituye un componente esencial para garantizar la confiabilidad y estabilidad de los sistemas. En aplicaciones médicas, donde los procesos de inferencia pueden involucrar modelos de deep learning y grandes volúmenes de datos, el monitoreo de métricas operativas resulta fundamental para evaluar el desempeño del sistema.

Herramientas como Prometheus y Grafana han sido ampliamente utilizadas para el monitoreo de sistemas distribuidos, permitiendo recolectar métricas relacionadas con latencia, uso de recursos computacionales y tasas de error. Sin embargo, investigaciones señalan que estas herramientas fueron diseñadas para aplicaciones generales no contemplan métricas específicas asociadas al análisis de imágenes médicas o a la ejecución de modelos de inteligencia artificial (Gomes et al., 2025; Li et al., 2022).

Por esta razón, estudios recientes han demostrado la necesidad de desarrollar mecanismos

de observabilidad especializados que permitan evaluar aspectos como la latencia de inferencia, la consistencia de resultados entre modelos y el comportamiento de los pipelines de procesamiento en aplicaciones médicas basadas en inteligencia artificial.

Síntesis del estado del arte:

A partir del análisis de las investigaciones revisadas, se observa que el desarrollo de sistemas de inteligencia artificial para aplicaciones médicas requiere la combinación de múltiples enfoques tecnológicos, incluyendo arquitecturas basadas en microservicios, infraestructuras cloud-native, estándares de interoperabilidad clínica y mecanismos avanzados de observabilidad.

Si bien existen diversas propuestas orientadas al despliegue de modelos de inteligencia artificial en entornos clínicos, aún persisten desafíos relacionados con la integración eficiente de estos modelos dentro de infraestructuras hospitalarias distribuidas, así como con la evaluación del desempeño de los sistemas en términos de latencia, consumo de recursos y escalabilidad.

La siguiente tabla presenta una comparación entre investigaciones relacionadas y el presente trabajo, identificando las dimensiones tecnológicas y metodológicas que distinguen cada propuesta.

Tabla 1
Comparación del presente trabajo con investigaciones relacionadas.

Trabajo	Arquitectura	Evaluación	Modelos	Contenerización	Métricas operativas	Seguridad
MLXOps4Medic (Huang & Liu, 2025)	Microservicios	Funcional	Múltiples	Sí	Parcial	No reportada
PACS-AI (Theriault-Lauzier et al., 2024)	Modular clínica	Clínica	Específicos	No especificado	No	Sí

Cloud DICOM (Brundage et al., 2022)	Cloud-native eventos	Funcional	N/A	Sí	No	No
Efficient nnU-Net (Magadza & Viriri, 2023)	Monolítica	Diagnóstica (Dice)	nnU-Net	No	No	No
Presente trabajo	Microservicios multicapa	Computacional	TF + nnU-Net	Sí (Docker)	Sí (latencia, GPU, RAM, errores)	Sí (API Key, cifrado, rate limiting)

Nota: La diferenciación principal del presente trabajo radica en la combinación de evaluación operativa completa, soporte multi-framework y mecanismos de seguridad integrados en un sistema contenerizado.

2.2. Marco teórico referencial

Plataformas de Model Serving

Los ecosistemas de plataformas para servir modelos en producción han evolucionado en gran medida con la adopción de prácticas de MLOps que integran principios de DevOps y exigencias específicas de Aprendizaje Automático (Machine Learning, ML), donde las prácticas de implementación de pipelines, Integración Continua (CI), Despliegue Continuo (CD), y las prácticas de monitoreo de modelos son las que se ejecutan sobre modelos en un contexto distribuido. Plataformas como TensorFlow Serving, MLflow y sistemas Kubernetes-native como Seldon Core se han estudiado recientemente dentro del dominio de MLOps, donde las exigencias de MLOps contemplan soluciones automáticas para el versionado de modelos, batching dinámico, escalabilidad horizontal y despliegues de microservicios desacoplados que son críticas para aplicaciones médicas que demandan alta disponibilidad, baja latencia y reproducibilidad de la inferencia (Garg et al., 2022; Panchal et al., 2023).

La separación de módulos entre la validación y la inferencia disminuye riesgos operativos, al favorecer la mantenibilidad de los modelos clínicos; la integración de pipelines automáticos asegura que los updates de los modelos pueden implementarse sin introducción de tiempos de mejora en los servicios críticos de salud digital, así como también cumplir con los requerimientos regulatorios de disponibilidad y confiabilidad. Junto a ello, el trabajo de MLOps nos muestra que la combinación de serving + tracking de experimentos y observabilidad distribuida en arquitecturas cloud-native para la implementación de pipelines resilientes extensibles al uso adecuado de recursos computacionales y la integración sencilla con microservicios dedicados a tareas diversas como la de procesamiento de imágenes médicas, segmentación de tumores o análisis cuantitativo de resultados.

En resumen se puede decir que la implementación de arquitecturas de microservicios para model serving basados en las propuestas de MLOps supone para los autores una figura crítica para cerrar la brecha que existe entre la lucha de las investigaciones desarrolladas en el campo de la IA médica y la implementación de las aplicaciones clínicas operativas, pues estas arquitecturas de microservicios garantizan la reproducibilidad, escalabilidad y confiabilidad que exige la infraestructura hospitalaria moderna (Hewage & Meedeniya, 2022).

Arquitecturas Cloud-Native para IA Médica

Las arquitecturas cloud-native han convertido el desarrollo de aplicaciones médicas basadas en IA al admitir flexibilidad computacional, aislamiento por medio de contenedores y desplegar en entornos híbridos. Análisis recientes han revelado que adoptar infraestructuras que se orientan a eventos y microservicios agilizan el procesamiento escalable de datos médicos en formatos estructurados como DICOM y FHIR (Tang et al., 2023).

Integrar IA en ambientes hospitalarios necesita hacer uso de plataformas que tengan la capacidad de ejecutar modelos de deep learning con el objetivo de garantizar el cumplimiento normativo, la privacidad y la trazabilidad de datos. Recientes investigaciones acerca de la integración de IA en estos entornos han manifestado que una arquitectura modular desacoplada es esencial debido a que asegura la interoperabilidad

con PACS y también con sistemas de historia clínica electrónica (Garg et al., 2022; Nopour, 2026).

Hacer uso de infraestructuras cloud que se especializan en salud digital es determinado como una estrategia para adaptar pipelines de inferencia distribuidos, teniendo como fin asegurar que esté disponible, que sea escalable, y que tenga gobernanza de datos en aplicaciones médicas críticas (Tang et al., 2023).

Estándares de Interoperabilidad Médica

Para que se pueda establecer sistemas con inteligencia artificial en entornos hospitalarios, la interoperabilidad es esencial puesto que implementar estándares como DICOM, permiten exponer objetos médicos mediante APIs RESTful integrando arquitecturas cloud-native y microservicios (Usman et al., 2022).

Un estándar que está establecido para el intercambio de información clínica estructurada es FHIR R4, su punto fuerte es que ofrece acceso mediante APIs HTTP y promueve la interoperabilidad.

Estudios han podido manifestar que hacer una combinación entre DICOMweb y FHIR R4, acondicionan ambientes donde modelos de inteligencia artificial se integran en flujos clínicos interoperables sin comprometer estándares regulatorios.

Ahora en cuanto a la integración en entornos hospitalarios, investigaciones destacan que los perfiles “Integrating the Healthcare Enterprise” (IHE) complementan estándares definiendo flujos de interoperabilidad que permite reducir imprecisiones al momento de implementarlo, y garantizando que tenga consistencia en entornos multicéntricos. (Faseeha et al., 2025).

Herramientas de Observabilidad y Monitoring

Uno de los componentes esenciales en arquitecturas de microservicios es la observabilidad, porque garantiza confiabilidad, trazabilidad, y desempeño en aplicaciones médicas críticas. Recientes estudios en sistemas distribuidos han señalado

que métricas de latencia, tasas de errores, y consistencia son determinantes para evaluar robustez operativa en pipelines de segmentación médica (Panchal et al., 2023).

Con el fin de diagnosticar fallos dentro de sistemas clínicos y detectar cuellos de botella a nivel de arquitecturas desacopladas, el monitoreo en series temporales es fundamental. Dentro de ingeniería de software aplicada a la salud digital, integrar frameworks de observabilidad reduce significativamente tiempos de recuperación ante fallos, y mejora significativamente la confiabilidad en sistemas que se basan en inteligencia artificial (Li et al., 2021).

CAPÍTULO III. METODOLOGÍA

3.1 Tipo y nivel de investigación

Este trabajo está desarrollado bajo un enfoque de investigación aplicada, que se orienta al diseño y a la implementación de soluciones tecnológicas con el fin de analizar imágenes utilizando modelos de inteligencia artificial.

El estudio se enmarca dentro del nivel de investigación experimental, proponiendo el desarrollo de una arquitectura en microservicios, para después evaluar el desempeño del sistema con la utilización de métricas computacionales durante la ejecución de procesos de segmentación de imágenes médicas.

Además, se incluye un componente tecnológico, ya que involucra el desarrollo de software especializado, integrando modelos de deep learning preentrenados, y la implementación de una infraestructura desplegada con métodos de contenerización.

Se busca evaluar el desempeño del sistema desarrollado con el uso de métricas para medir la latencia durante la inferencia, y la utilización de recursos como la memoria RAM y de procesamiento gráficos (GPU - CPU).

3.2 Diseño metodológico

El diseño se basa en implementar y evaluar una arquitectura a nivel de microservicio para procesar imágenes médicas con el uso de modelos preentrenados para una segmentación automatizada de tumores cerebrales.

El proceso metodológico se ha estructurado en fases, las cuales se detallan a continuación:

1. Revisión bibliográfica.- Se ha realizado un análisis científico relacionado con arquitectura de microservicios aplicadas a la inferencia de modelos de inteligencia artificial en salud, así como de estudios de sistemas que integren la segmentación automatizada de tumores cerebrales usando modelos preentrenados de deep learning.
2. Diseño de la arquitectura del sistema.- Se definió la arquitectura de microservicios porque permite separar responsabilidades en distintos componentes del sistema, como servicios de segmentación, clasificación de imágenes y gestión de solicitudes.

3. Implementación de microservicios.- Se desarrollaron microservicios independientes que ejecutan procesos de inferencia para la segmentación de tumores cerebrales en imágenes de resonancia magnética.
4. Integración con sistemas de procesamiento de imágenes médicas.- La arquitectura de microservicios permite el procesamiento de imágenes médicas en formato NIfTI / DICOM. Esto permite una fácil integración con sistemas clínicos ya implementados.
5. Evaluación del sistema.- Se realizaron pruebas para evaluar el comportamiento y desempeño del sistema, mediante el uso de métricas relacionadas con el tiempo de procesamiento, el uso de los recursos de hardware y estabilidad operativa.

3.3 Dataset utilizado

Para el entrenamiento y evaluación de los modelos se emplearon imágenes del Brain Tumor Segmentation Challenge (BraTS 2023), específicamente del sub-reto de segmentación de Glioma de Grado Adulto (GLI – Adult Glioma), que es un conjunto de datos de referencia ampliamente utilizado en la comunidad científica dedicada a la segmentación de tumores cerebrales. Este dataset fue descargado desde la plataforma oficial Synapse (synapse.org, proyecto syn51156910), repositorio oficial del challenge MICCAI 2023.

El dataset incluye imágenes de resonancia magnética multimodal T1, T1 con contraste, T2 y FLAIR acompañadas de máscaras de segmentación anotadas por expertos clínicos. Contar con etiquetas generadas por especialistas es un punto importante, ya que aporta consistencia a los datos y da solidez a cualquier comparación con trabajos anteriores.

Casos utilizados en las pruebas de inferencia:

Para las pruebas de desempeño del microservicio se emplearon dos conjuntos de casos del dataset BraTS 2023. En el entorno local (equipos ASUS y ACER) se seleccionaron 3 casos representativos del conjunto de validación (BraTS-GLI-00000, 00002 y 00003), sobre los cuales se realizaron 10 repeticiones consecutivas para evaluar la estabilidad y variabilidad de la latencia bajo condiciones controladas. Para las pruebas en el servidor SPARK se utilizaron 30 casos seleccionados aleatoriamente mediante semilla fija

(seed=42), lo que garantiza reproducibilidad y proporciona mayor representatividad estadística.

Las 10 ejecuciones registradas en las tablas de resultados corresponden a repeticiones consecutivas sobre los mismos casos, estrategia adoptada para obtener mediciones estadísticamente estables de latencia y consumo de recursos, descartando variaciones atípicas producidas por la carga inicial del modelo en memoria.

Características del dataset:

Tabla 2
Características del dataset BraTS 2023 utilizado.

Característica	Detalle
Challenge	ASNR-MICCAI BraTS 2023 — Adult Glioma (GLI)
Fuente oficial	Synapse — synapse.org (syn51156910)
Modalidades de resonancia	T1, T1c (T1 con contraste), T2, FLAIR
Tipo de tumor	Glioma de grado adulto
Formato de imágenes	NIfTI (.nii.gz) — volúmenes tridimensionales
Tamaño promedio por volumen	Entre 2.37 MB y 2.70 MB
Repeticiones por condición	10 ejecuciones consecutivas por configuración

Es importante señalar que el uso del dataset en esta investigación se limitó a la etapa de inferencia. Los modelos empleados fueron utilizados con sus pesos preentrenados disponibles, entrenados sobre los datos de BraTS 2023. No se realizó una división train/test en este trabajo, ni se validó los modelos, dado que el alcance es la evaluación computacional del microservicio, no la validación diagnóstica de los modelos.

Pruebas masivas con conjuntos de datos del dataset de BraTS 2023:

Se utilizaron imágenes con formato NIfTI (.nii.gz) del dataset BraTS 2023, modalidad T1c (T1 con contraste), correspondientes a gliomas de alto grado (GLI).

Tabla 3
Datos BraTS 2023

Parámetro	Valor
Dataset	BraTS 2023 — 44 imágenes disponibles
Imágenes por ejecución	30 (selección aleatoria con semilla 42)
Runs de calentamiento	3 (descartados del análisis)
Runs válidos	30 por configuración (n=30, poder estadístico mínimo)
Modelo evaluado	nnU-Net V2 (PyTorch) — 30,787,604 parámetros
Hardware	GPU NVIDIA GB10 (cuda:0
Timeout por request	300 segundos
Semilla aleatoria	42 (para reproducibilidad)
Escenarios	Local (localhost:3021) y Externo (172.16.105.181:3021)

3.4 Modelos utilizados

Para la evaluación del desempeño computacional del microservicio se emplearon dos modelos de segmentación de imágenes médicas preentrenados, seleccionados por su diferencia en complejidad arquitectónica, lo que permite evaluar el comportamiento del sistema bajo distintos niveles de demanda computacional. En ambos casos se utilizaron los pesos preentrenados disponibles, sin realizar ningún proceso de reentrenamiento, dado que el propósito de su integración es generar carga de inferencia real sobre el microservicio y no evaluar la calidad diagnóstica de los modelos.

UNet 3D MultiScale (TensorFlow)

El primer modelo corresponde a una arquitectura UNet tridimensional con módulos multiescala, implementada en TensorFlow y almacenada en formato .h5. Cuenta con aproximadamente 5.64 millones de parámetros entrenables, lo que representa una demanda computacional moderada.

nnU-Net V2 (PyTorch)

El segundo modelo corresponde a la arquitectura nnU-Net en su versión 2, implementada en PyTorch y almacenada en formato .pth. Con aproximadamente 30.7 millones de parámetros en la configuración evaluada y más de 88 millones de operaciones tensoriales por inferencia, representa una carga computacional significativamente mayor que el modelo anterior.

La selección de estos dos modelos responde a la necesidad de contrastar el comportamiento del microservicio ante perfiles de carga heterogéneos: un modelo ligero portable a entornos sin GPU, y un modelo de alta demanda representativo de arquitecturas de referencia en segmentación biomédica. Esta combinación permite obtener una caracterización más completa del sistema en términos de latencia, consumo de memoria y utilización de recursos gráficos, cubriendo un espectro amplio de escenarios de despliegue clínico.

3.5 Variables y métricas de evaluación

Con el objetivo de evaluar el desempeño del sistema desarrollado se han definido las siguientes variables de estudio:

- **Latencia de inferencia**

  Se refiere al tiempo total que se requiere para procesar una petición desde el momento que el sistema recibe la solicitud hasta que se genera el resultado final. Se expresa en segundos y evalúa la eficiencia del sistema durante la ejecución.

- **Uso de memoria**

  Corresponde a la cantidad de RAM y VRAM usada durante la ejecución de la inferencia de segmentación. Se tomó en cuenta esta métrica con la finalidad de su análisis y su auditoría para evaluar su desempeño en entornos de producción.

- **Utilización de recursos de procesamiento CPU y GPU**

  Corresponde al porcentaje de utilización de unidades de procesamiento (CPU o GPU) durante la ejecución del proceso de inferencia de imágenes médicas. Esta métrica permite una evaluación con respecto al hardware disponible.

- **Tasa de errores**

  Se define como la proporción de solicitudes que no han podido ser completadas con éxito debido a fallos durante la ejecución del sistema. Permite evaluar la estabilidad del microservicio en un ambiente con múltiples peticiones simultáneas.

Tabla 4
Variables y descripción de las métricas utilizadas.

Variable	Tipo	Métrica	Unidad	Descripción
Latencia de inferencia	Dependiente	Tiempo de procesamiento	Segundos	Tiempo requerido para procesar una solicitud completa de segmentación
Uso de memoria	Dependiente	Consumo de RAM / VRAM	MB	Cantidad de memoria RAM / VRAM utilizada durante la ejecución
Utilización de CPU/GPU	Dependiente	Porcentaje de uso	%	Porcentaje de uso del procesador durante la inferencia
Tasa de errores	Dependiente	Solicitudes fallidas	%	Proporción de solicitudes que fallan durante el procesamiento

3.6 Arquitectura de sistema desarrollado

La investigación está enfocada en el diseño y la integración del microservicio con la finalidad de segmentar imágenes médicas en un ambiente de arquitectura distribuida. Para integrar este sistema de forma estructurada, escalable y mantenible se optó por un enfoque que permita ejecutar modelos de deep learning.

La adopción de una arquitectura basada en microservicios implicó un mayor nivel de complejidad en comparación con enfoques monolíticos, debido a la necesidad de definir responsabilidades específicas para cada componente del sistema, entre uno de ellos: comunicación de servicios y métodos de despliegue. Durante la planificación implicó más complejidad, sin embargo, el sistema es modular, permitiendo que cada módulo pueda ser desarrollado, probado y desplegado independientemente.

Durante el desarrollo del sistema fue necesario tomar diversas decisiones técnicas relacionadas con la integración del modelo y la lógica del servicio. Algunas de ellas surgieron al momento de integrar el modelo de aprendizaje profundo con la lógica del servicio, donde la forma en que se gestionan los datos de entrada y salida influye directamente en el rendimiento del sistema. Estas decisiones se fueron documentando a medida que avanzaba el proyecto, con el fin de mantener coherencia entre los distintos módulos y facilitar futuras modificaciones.

El desarrollo siguió principios de arquitectura limpia y separación de responsabilidades, apoyándose también en buenas prácticas de DevOps. Esto se tradujo en decisiones concretas: cómo estructurar el código internamente, qué herramienta de contenerización usar, cómo definir las interfaces de comunicación y de qué manera configurar el proceso de despliegue. Cada una de estas decisiones buscó que el sistema fuera mantenible y coherente con los estándares actuales de desarrollo de software.

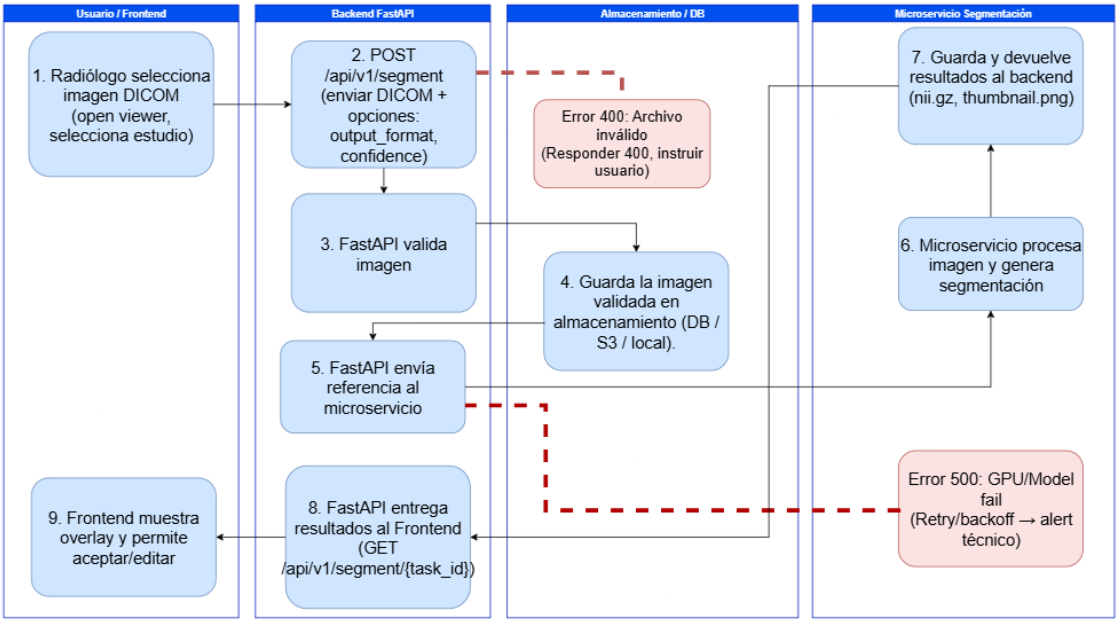
3.7 Flujo de trabajo del sistema

Previo a la implementación del sistema, se definió el flujo de procesamiento de la información con el fin de establecer la secuencia de interacción entre los diversos componentes del sistema. Una de las primeras tareas fue trazar el recorrido completo que sigue una imagen médica desde que el médico la selecciona hasta que el sistema devuelve los resultados de segmentación.

Para documentar este recorrido se construyó un User Journey Diagram con la ayuda de la herramienta draw.io, que resultó útil no solo para visualizar el flujo, sino también para identificar puntos críticos donde podían ocurrir fallos. El diagrama dejó en evidencia que el proceso no es lineal en todos los casos: hay rutas alternativas dependiendo de si la imagen es válida o si el microservicio responde correctamente.

Figura 1

Issue board de la arquitectura general del sistema (Flujo de trabajo), mostrando de forma detallada la secuencia de interacción entre los distintos componentes involucrados en el procesamiento de imágenes dentro del microservicio de segmentación.



Nota. Elaboración propia.

El sistema propuesto tiene dos modalidades operativas bien definidas: segmentación individual y segmentación por lotes. La segmentación individual, centrada en procesar una sola imagen por cada solicitud, mientras que la modalidad por lotes permite enviar y procesar varias imágenes al mismo tiempo, adaptándose a situaciones donde se requiere manejar un mayor volumen de datos.

Metodológicamente, el flujo inicia cuando la imagen llega a través de la API, pasando primero por un proceso de validación y preprocesamiento. Luego, se envía al motor de inferencia para ejecutar el modelo de segmentación, y, finalmente, los resultados se gestionan mediante los mecanismos de almacenamiento y transferencia definidos en el sistema. Todo este flujo se diseñó y documentó a partir de los issue boards del proyecto, garantizando que su implementación fuera coherente y trazable.

El sistema desarrollado presenta dos modalidades operativas:

- La primera es la segmentación individual, pensada para los casos en que el usuario necesita procesar un único estudio de forma inmediata.

- La segunda es la segmentación por lotes, que fue incorporada para responder a escenarios con mayor volumen de procesamiento. En entornos hospitalarios con alta demanda, no tiene sentido procesar imagen por imagen si se pueden enviar varias al mismo tiempo. Esta modalidad permite cargar múltiples archivos simultáneamente, lo que reduce los tiempos de espera cuando el sistema está bajo carga sostenida.

3.8 Arquitectura general del microservicio

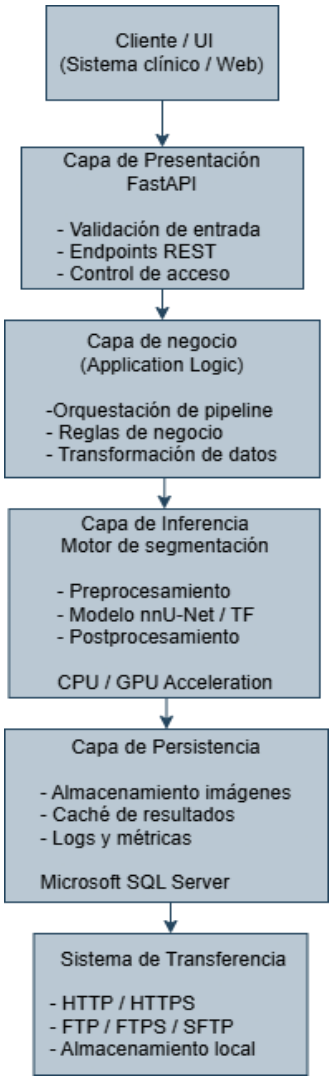
La definición de la arquitectura constituye una de las decisiones centrales del proyecto, dado que determina la organización del código, los mecanismos de conexión entre componentes y la facilidad de modificación futura del sistema. Se optó por un enfoque multicapa, tomando como punto de partida el issue board de arquitectura general del sistema, donde ya estaban esbozadas algunas decisiones previas del equipo.

La arquitectura quedó organizada en cuatro capas principales:

- Capa de presentación
- Capa de negocio
- Capa de inferencia
- Capa de persistencia

La separación de responsabilidades en capas independientes permite aislar el impacto de modificaciones futuras, reduciendo el riesgo de introducir errores en cascada ante cambios en componentes individuales del sistema. Al distribuir las responsabilidades, es posible trabajar en cada parte del sistema de forma más ordenada y con menos riesgo de errores.

Figura 2
Arquitectura general del microservicio de segmentación médica.



Nota. Arquitectura multicapa del microservicio desarrollado para la segmentación de imágenes médicas. Se ilustran las capas de presentación, lógica de negocio, inferencia y persistencia, así como los mecanismos de transferencia y almacenamiento de resultados. Elaboración propia.

3.8.1 Capa de presentación (API)

La capa de presentación es, en términos prácticos, la puerta de entrada al microservicio: todo lo que viene del exterior pasa primero por aquí. Se implementó como una API RESTful, una elección que resultó conveniente porque simplifica enormemente la integración con el frontend y con el resto de servicios del sistema, sin tener que recurrir a protocolos más complejos. Su rol es recibir las solicitudes de segmentación, exponer

los endpoints definidos y actuar como intermediaria entre los clientes —ya sean usuarios o sistemas externos y la lógica interna del microservicio.

Entre sus responsabilidades más importantes está la validación de las solicitudes entrantes: antes de que cualquier dato avance en el flujo, esta capa verifica que el tipo de archivo sea el correcto, que los tamaños estén dentro de lo permitido y que el formato sea compatible. Además de eso, se ocupa de estructurar las respuestas y de manejar los errores de manera uniforme, algo que se vuelve especialmente relevante cuando el sistema trabaja tanto en modo síncrono como asíncrono.

3.8.2 Capa de negocio (Lógica de aplicación)

Si la capa de presentación es la puerta de entrada, la capa de negocio es donde realmente se toman las decisiones. Es aquí donde se establece el orden en que se ejecutan las etapas del pipeline, bajo qué condiciones puede avanzarse de una etapa a la siguiente y cómo se responde ante los casos que se salen de lo esperado.

Sus responsabilidades son bastante concretas: valida que se cumplan las reglas propias del sistema, transforma los datos tanto en la entrada como en la salida, y mantiene la comunicación entre la capa de inferencia, la de persistencia y el sistema de transferencia. Su diseño metodológico garantiza que el comportamiento del sistema sea coherente, controlado y alineado con los objetivos del proyecto.

La capa de negocio encapsula las reglas y procesos principales del sistema, tales como:

- **Orquestación de los flujos de procesamiento de imágenes**

  Consiste en la coordinación secuencial y lógica de las distintas etapas del procesamiento de imágenes, definiendo el orden de ejecución desde la recepción de la imagen hasta la obtención de los resultados, asegurando que cada fase del pipeline se ejecute de manera controlada y consistente.

- **Validación de reglas de negocio**

  Implica la verificación de que las solicitudes y los datos recibidos cumplan con los criterios y restricciones definidos por el sistema, tales como formatos permitidos, parámetros de ejecución y políticas de uso, antes de iniciar el proceso

de segmentación.

- **Transformación de datos de entrada y salida**

  Se refiere a la adaptación de los datos de entrada al formato requerido por los modelos de inferencia, así como a la estructuración y normalización de los resultados generados, facilitando su almacenamiento, transferencia y consumo por otros componentes del sistema.

- **Coordinación entre la capa de inferencia y los sistemas de persistencia y transferencia**

  Comprende la gestión de la comunicación entre el motor de inferencia, los mecanismos de almacenamiento y los servicios de transferencia, garantizando que los resultados sean correctamente registrados, almacenados y, cuando corresponda, enviados a sistemas externos de forma confiable.

3.8.3 Capa de inferencia (Modelo de segmentación)

La capa de inferencia constituye uno de los componentes más críticos del sistema. Es el punto donde el modelo de inteligencia artificial entra en acción, y por eso buena parte de las decisiones de diseño del microservicio giraron en torno a ella. El núcleo de esta capa son modelos de segmentación basados en aprendizaje profundo, principalmente bajo la arquitectura nnU-Net en su variante tridimensional (3D), que se ha ganado un lugar destacado en tareas de segmentación biomédica por sus resultados consistentemente buenos.

El sistema desarrollo admite el uso de múltiples frameworks de deep learning por ejemplo, modelos entrenados tanto en TensorFlow (formato .h5) como en PyTorch (formato .pth), lo que da bastante flexibilidad a la hora de incorporar o cambiar arquitecturas según evolucionen las necesidades del proyecto.

Se implementó un mecanismo de carga dinámica de modelos, lo que permite la utilización de distintas versiones del modelo sin interrumpir el sistema. Este mecanismo permite que

el sistema trabaje con distintas versiones del modelo sin interrupciones de funcionamiento, dependiendo de la configuración o de lo que el usuario necesite en cada momento.

El flujo que sigue una imagen dentro de esta capa tiene tres momentos bien diferenciados. Primero pasa por un preprocesamiento que la adapta a lo que el modelo espera recibir: ajuste de tamaño, normalización de valores y conversión de formato. Después viene la inferencia propiamente dicha, que se ejecuta aprovechando tanto CPU como GPU para no desperdiciar recursos computacionales. Y al final, los resultados pasan por una etapa de postprocesamiento que los convierte en salidas legibles y útiles, tanto para el resto del sistema como para quien esté al otro lado.

Incluye:

- **Carga y gestión dinámica de modelos:**
  Una de las decisiones más prácticas en el diseño de esta capa fue permitir que los modelos se carguen en tiempo de ejecución. Esto significa que no hace falta reiniciar ni reconfigurar el sistema para cambiar de modelo o cargar una versión distinta del mismo: según la configuración activa o lo que el proceso de análisis requiera en ese momento, el sistema selecciona, actualiza o reutiliza el modelo correspondiente sin interrupciones.

- **Preprocesamiento de imágenes de entrada:**

  Antes de que una imagen llegue al modelo, hay un trabajo previo que no es menor. En esta etapa se aplican distintas operaciones sobre los datos: redimensionamiento, normalización de intensidades y conversión de formato, entre otras. El propósito es bastante directo: adaptar las imágenes a lo que el modelo espera recibir, respetando sus características y limitaciones, para que el procesamiento posterior pueda llevarse a cabo sin inconvenientes.

- **Ejecución eficiente de la inferencia, con soporte para CPU y GPU:**

  Una vez que los datos están listos, la inferencia se ejecuta buscando aprovechar al máximo los recursos computacionales disponibles. El sistema admite su ejecución tanto en CPU como en GPU, lo que ayuda a reducir los tiempos de procesamiento

y mejora el rendimiento general.

- **Postprocesamiento de los resultados generados por el modelo:**

  Una vez que el modelo genera su salida, hay una etapa de postprocesamiento que convierte ese resultado en algo utilizable. Para ello, se generan máscaras segmentadas y, cuando es necesario, se aplican ajustes de umbral o refinamientos en las regiones detectadas, lo que facilita su almacenamiento, visualización y transferencia posterior.

3.8.4 Capa de persistencia

Durante la fase de diseño se identificó la necesidad de definir un mecanismo de persistencia que permitiera conservar los resultados generados por el modelo para su posterior recuperación, auditoría y reanudación en caso de interrupción del proceso. No basta con generarlos: hay que guardarlos de forma que puedan recuperarse después, ya sea porque el usuario quiere revisarlos más tarde, porque el sistema necesita auditarlos, o porque una falla interrumpió el proceso y hay que retomarlo. De eso se encarga la capa de persistencia.

Su función más directa es guardar localmente de las imágenes segmentadas, así como la implementación de mecanismos de caché que evita reprocesar imágenes que ya fueron segmentadas anteriormente.

Además, cabe recalcar que esta capa de persistencia incorpora los siguientes mecanismos:

- Almacenamiento local de imágenes segmentadas.
- Caché de predicciones para evitar reprocesamientos innecesarios.
- Registro de logs y métricas de ejecución.
- Soporte para auditoría y análisis posterior.

3.9 Componentes principales del sistema

3.9.1 Gestor de modelos

Uno de los componentes esenciales, debido a que se encarga de administrar las distintas versiones del modelo de segmentación utilizadas por el sistema. Este componente facilita la selección dinámica del modelo adecuado, permitiendo actualizar o cambiar modelos sin afectar el funcionamiento general del microservicio.

Sus funciones son:

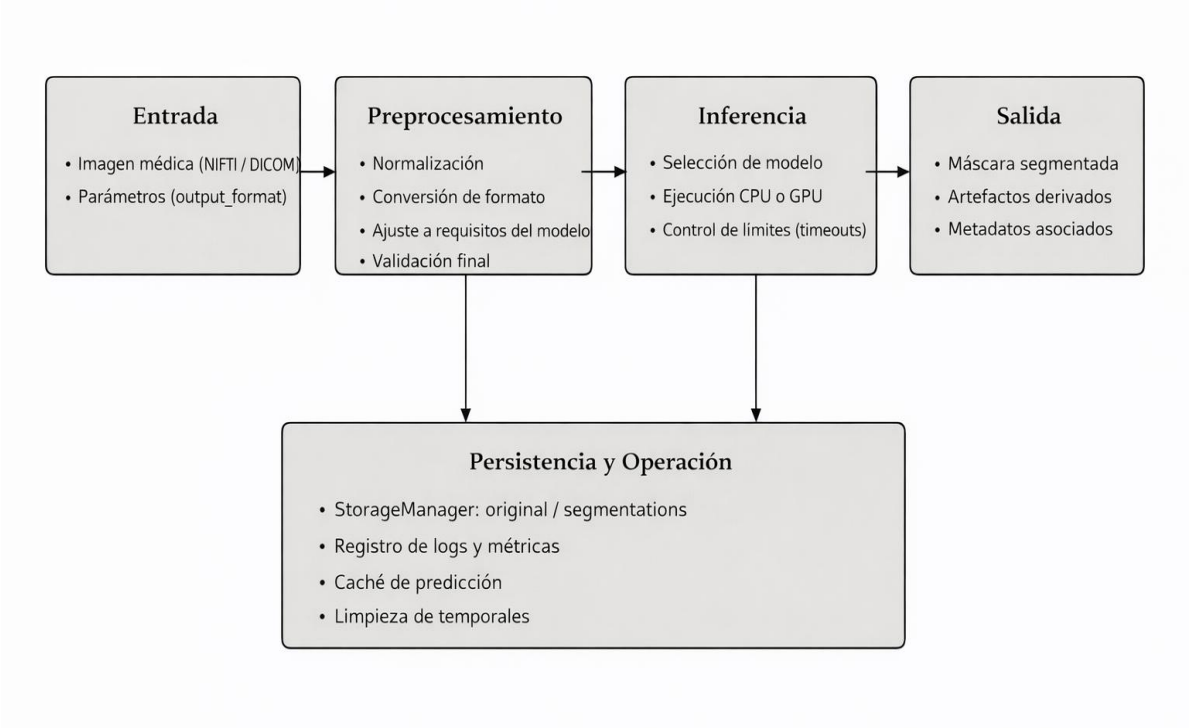
- Selección dinámica de versiones de modelo.
    - Carga de modelos entrenados (formatos Tensorflow (.h5) y nnUNet (.pth)).
    - Implementación de caché de modelos en memoria para reducir tiempos de carga.
    - Validación de compatibilidad entre modelos y configuraciones del sistema.

Los modelos entrenados se gestionan en diferentes formatos, e incorporan mecanismos de almacenamiento en caché que reducen los tiempos de carga y permiten validar su compatibilidad con la configuración del sistema. Esto garantiza un uso estable y flexible de los modelos de aprendizaje automático dentro del microservicio.

3.9.2 Pipeline de procesamiento

El pipeline de procesamiento define las etapas por las cuales pasa una imagen desde su ingreso al sistema hasta la generación del resultado final. El flujo de datos se estructura en tres fases principales: **preprocesamiento**, **inferencia** y **postprocesamiento**, las cuales permiten organizar y controlar de forma sistemática el procesamiento de imágenes médicas dentro del microservicio.

Figura 3
Arquitectura del pipeline de procesamiento del microservicio de segmentación médica



Nota. Elaboración propia.

a) Preprocesamiento

La fase de preprocesamiento tiene como objetivo adaptar las imágenes de entrada a los requisitos técnicos del modelo de segmentación, asegurando compatibilidad, estabilidad y uniformidad en los datos procesados. En esta etapa se aplica las siguientes operaciones:

La primera es la normalización. Lo que se hace aquí es ajustar los valores de intensidad de los píxeles a un rango estándar. Este proceso reduce la variaciones entre imágenes y contribuye a mejorar la estabilidad del modelo. Después viene el redimensionamiento, donde las imágenes se escalan a las dimensiones que el modelo espera recibir. Sin este paso, habría incompatibilidades que harían inviable el procesamiento posterior.

Por último se aplica la conversión de formato. Las imágenes se transforman al formato compatible con el pipeline y el modelo, lo que garantiza que todos los datos de entrada lleguen de manera uniforme y sin inconsistencias que puedan interrumpir la ejecución.

b) Inferencia

Con la imagen ya preparada, comienza la parte central del proceso. La inferencia es la etapa donde el modelo de segmentación se aplica sobre las imágenes procesadas y genera las predicciones de las regiones de interés. No es un paso trivial desde el punto de vista computacional, y por eso su diseño estuvo orientado desde el principio a optimizar el rendimiento y gestionar los recursos de forma eficiente. En esta etapa incluye las siguientes ejecuciones.

- ***La ejecución del modelo de segmentación:***

  En esta capa se ejecuta el modelo encargado de generar las predicciones de segmentación, trabaja sobre la imagen y produce las predicciones correspondientes a las regiones segmentadas.

- ***Procesamiento por lotes:***

  El sistema permite procesar varias imágenes al mismo tiempo, lo que optimiza el rendimiento y reduce la latencia, especialmente en escenarios con un alto volumen de datos.

- ***Optimización de recursos CPU/GPU:***

  La inferencia se ajusta de forma automática a los recursos que tenga disponibles el sistema. Si existe una GPU, aprovecha su capacidad para acelerar el procesamiento; en cambio, si no hay hardware especializado, se ejecuta sin problemas utilizando únicamente la CPU.

- ***Control de timeouts y límites de ejecución:***

  Cada proceso de inferencia cuenta con límites definidos de tiempo y uso de recursos. Esto evita que el sistema se bloquee y ayuda a que todo siga funcionando con normalidad, incluso cuando la carga de trabajo es alta.

c) Postprocesamiento

La fase de postprocesamiento se ocupa de tomar las salidas crudas del modelo y convertirlas en resultados organizados y listos para ser utilizados por el sistema. Es en

este punto donde la información se prepara y se le da la estructura necesaria. Para lograrlo, se aplican las siguientes técnicas:

- ***Umbralización de resultados:***

  Se aplican los umbrales sobre las salidas del modelo para definir con claridad las regiones segmentadas, facilitando la interpretación de los resultados.

- ***Generación de máscaras de segmentación:***

  Los resultados son transformados en máscaras binarias o multicategorías que representan las áreas identificadas por el modelo.

- ***Filtrado de ruido:***

  Se eliminan artefactos o regiones irrelevantes mediante técnicas de filtrado, mejorando la calidad visual y analítica de la segmentación.

- ***Cálculo de métricas asociadas a la inferencia:***

  Incluye la generación y registro de métricas técnicas derivadas del proceso de inferencia, tales como tiempos de ejecución y estadísticas de salida del modelo, las cuales son utilizadas posteriormente para el análisis del desempeño del sistema.

3.9.3 Sistema de entrada y salida

El sistema de entrada y salida gestiona los archivos que ingresan y salen del microservicio, soportando:

- Múltiples formatos médicos e imagenológicos (NIfTI, DICOM, entre otros).
- Validación de tipos MIME y tamaños máximos.
- Serialización y compresión de resultados.
- Preparación de respuestas optimizadas para el cliente.

3.10 Endpoints del microservicio

El microservicio expone un conjunto de endpoints diseñados para cubrir distintos escenarios de uso, permitiendo la segmentación individual, por lotes y de forma asíncrona, así como la consulta de resultados, modelos disponibles y el estado general del servicio. Estos endpoints constituyen la interfaz principal entre el sistema y sus consumidores.

- POST /api/v1/predict/nnUNet: Segmentación de una imagen individual.
- POST /api/v1/predict/nnUNet/batch: Segmentación de múltiples imágenes.
- GET /api/v1/health: Verificación del estado del servicio.

3.11 Formatos de respuesta

Una de las decisiones que facilita bastante la integración del sistema es que los resultados no se entregan en un único formato fijo. Dependiendo del contexto y de lo que necesite quien consume el servicio, es posible elegir entre diferentes opciones de salida, como por ejemplo máscaras binarias de segmentación o metadatos adicionales como nivel de confianza, tiempo de inferencia y clases detectadas.

3.12 Gestión de recursos y optimización

Para garantizar el rendimiento y la estabilidad del sistema, se implementaron diversas estrategias de optimización, entre las cuales se incluyen:

- Pool de workers para procesamiento concurrente.
- Sistema de colas para peticiones asíncronas.
- Versionado de modelos para pruebas A/B.
- Caché inteligente de resultados frecuentes.
- Rate limiting para control de carga.
- Circuit breakers para prevenir fallos en cascada.

3.13 Monitoreo y observabilidad

Estas métricas fueron definidas con el objetivo de evaluar el comportamiento operativo del sistema, sin constituir una validación clínica del modelo. Por eso se incorporaron mecanismos de monitoreo desde etapas tempranas del desarrollo, con el fin de evaluar el comportamiento operativo en tiempo real. Es importante señalar que estas métricas no constituyen una validación clínica del modelo, sino que apuntan a entender su desempeño desde el punto de vista del sistema.

- ***Latencia de inferencia:***

  Esta métrica mide el tiempo requerido por el sistema para procesar una solicitud, desde que recibe una imagen hasta que devuelve la predicción. Es un indicador que no puede ignorarse, sobre todo en contextos donde los tiempos de respuesta importan, como ocurre en entornos clínicos asistidos por computadora.

- ***Throughput:***

  Otra métrica relevante es el throughput. A diferencia de la latencia, que mide la velocidad de una sola operación, esta métrica permite evaluar la capacidad del sistema para procesar múltiples solicitudes en condiciones de alta carga: ya sea porque llegan varias solicitudes simultáneas o porque debe mantener un ritmo de procesamiento sostenido. Con este valor es posible estimar qué tan eficiente resulta el sistema y anticipar cómo respondería ante una mayor demanda.

- ***Uso de memoria y Procesamiento CPU/GPU:***

  El uso de memoria y CPU/GPU fue otra variable que se decidió monitorear. Se llevó a cabo un análisis para conocer cuántos recursos computacionales consume el sistema durante la inferencia. Esta métrica permite evaluar la viabilidad del sistema en entornos con recursos limitados.

- ***Tasa de errores operativos:***

  Permite representar el porcentaje de solicitudes que fallan en relación con el total de peticiones que recibe el microservicio. Este indicador permite detectar posibles problemas, ya sea en la ejecución del pipeline, en la compatibilidad de los datos de entrada o incluso en la propia infraestructura donde se ejecuta el sistema.

3.14 Seguridad de los datos

La seguridad de la información es clave en los sistemas que procesan imágenes médicas, principalmente porque manejan datos sensibles de los pacientes. Para ello, se tomó la decisión que el microservicio desarrollado proteja la información de forma integral, aplicando medidas que garantizan su confidencialidad, integridad y disponibilidad durante todo el proceso.

- ***Api keys***

El control de acceso fue considerado como un requisito fundamental en el diseño del sistema. Los endpoints del sistema están protegidos mediante API Keys, lo que garantiza que solo los usuarios o servicios debidamente autorizados puedan interactuar con los recursos disponibles. Estas claves permiten restringir el acceso al sistema y facilitan la identificación de cada cliente que interactúa con los servicios, facilitando un seguimiento preciso de quién hace qué dentro del sistema.

- ***Validación entrada/salida***

Otro punto que se cuidó especialmente fue lo que entra y lo que sale del sistema. Se implementó una validación rigurosa de los archivos y datos recibidos, revisando el tipo de archivo, su tamaño, la estructura interna y el formato esperado. El objetivo es doble: por un lado, bloquear cualquier contenido malicioso o incompatible antes de que avance en el pipeline; por otro, asegurar que solo se procesen imágenes médicas que cumplan con los criterios definidos. Sin este control, la integridad de los datos y la estabilidad del sistema estarían en riesgo.

- ***Rate limiting***

El sistema incorpora mecanismos de rate limiting aplicados por usuario o por dirección IP. En términos prácticos, esto significa que hay un límite sobre cuántas solicitudes puede hacer un cliente en un período determinado. La razón es bastante directa: proteger el servicio frente a posibles abusos o ataques de denegación de servicio (DoS), y mantener un rendimiento estable incluso cuando la demanda se dispara, asegurando que los recursos del microservicio se mantengan protegidos y que el rendimiento se mantenga estable incluso cuando la carga es alta.

- ***Encriptación de datos***

Un aspecto relevante considerado en el diseño del sistema es la gestión de los datos, tanto cuando viajan por la red como cuando están en reposo. Para ello se implementó cifrado en ambos extremos del flujo: los datos de entrada se cifran antes de ser transmitidos, de modo que la información sensible permanezca protegida durante todo el trayecto. Una vez que el backend los recibe, se encarga de descifrarlos para que los modelos de segmentación puedan trabajar con ellos con normalidad. El frontend, por su parte, solo accede a resultados ya procesados y controlados, sin exposición directa a los datos clínicos originales.

Esta arquitectura responde a la necesidad de proteger la información sensible durante todo el flujo de procesamiento. Al evitar el almacenamiento innecesario de información y mantener el cifrado activo durante la transmisión, se reduce considerablemente el riesgo de filtraciones o accesos no autorizados, protegiendo así la confidencialidad de los datos de los pacientes.

Vistas en conjunto, todas las medidas descritas en esta sección: autenticación mediante API Keys, validación de entradas, rate limiting y cifrado, configuran un enfoque de seguridad óptimo. No se trata de medidas aisladas, sino de capas que se complementan entre sí para proteger la información sensible, mantener la integridad de las operaciones y garantizar que el sistema pueda sostenerse ante situaciones adversas. Al mismo tiempo, este diseño se alinea con las buenas prácticas que se esperan en microservicios orientados al análisis de imágenes médicas.

3.15 Tecnologías utilizadas

Cada herramienta que forma parte del microservicio fue elegida pensando en su estabilidad, su rendimiento y en qué tan bien encaja dentro de una arquitectura basada en microservicios. En conjunto, permiten integrar de forma coherente los distintos componentes del sistema: procesamiento de imágenes, segmentación automática e infraestructura de despliegue, todo bajo un mismo stack reconocido tanto en la industria del software como en proyectos de análisis de imágenes médicas.

- ***API: FastAPI Framework***

  Para la capa de interfaz RESTful se optó por FastAPI, y las razones fueron principalmente las siguientes:

1. Su arquitectura basada en ASGI le da un rendimiento notable.
2. Soporte nativo para programación asíncrona lo hace especialmente adecuado para un sistema que necesita manejar múltiples solicitudes sin bloquearse.
3. Validación automática de datos mediante tipado estático, lo que simplifica considerablemente el manejo de entradas y reduce la posibilidad de errores silenciosos.

- ***Modelos de segmentación automático – TensorFlow y nnU-Net***

  Para la segmentación se empleó la arquitectura nnU-Net 3D, complementada con un modelo implementado en TensorFlow. Esta integración permitió soportar múltiples frameworks dentro de un mismo pipeline de inferencia, favoreciendo la interoperabilidad y escalabilidad del sistema. La inclusión de ambos enfoques permitió evaluar distintos modelos dentro de un mismo pipeline de inferencia, favoreciendo la flexibilidad y reutilización del sistema.

- ***Procesamiento de imágenes y datos – OpenCV, Pillow, NumPy y Pandas***

  Estas librerías fueron integradas para la manipulación y transformación de imágenes médicas y datos asociados, permitiendo operaciones como redimensionamiento, normalización, conversión de formatos y manejo eficiente de arreglos numéricos, fundamentales en las etapas de preprocesamiento y postprocesamiento.

- ***Infraestructura: Docker, Kubernetes***

  Docker fue utilizado para encapsular el microservicio y sus dependencias, garantizando consistencia entre los entornos de desarrollo, prueba y despliegue.

Se considera que Kubernetes sirve para la orquestación de contenedores, facilitando la escalabilidad, la administración de recursos y la tolerancia a fallos en escenarios de mayor carga operativa.

3.16 Estrategias de despliegue

Para el despliegue del sistema se optó por contenedores Docker, con Dockerfiles diferenciados y optimizados según el entorno de ejecución: uno para CPU y otro para GPU. La gestión entre los entornos de desarrollo y producción se maneja a través de Docker Compose, que permite configurar redes y volúmenes de manera ordenada sin complicar innecesariamente la infraestructura.

Un aspecto que vale la pena destacar es el sistema dual de transferencia que se incorporó. Los resultados pueden guardarse localmente y, al mismo tiempo, enviarse de forma automática mediante protocolos como HTTP/HTTPS o FTP/FTPS/SFTP. Esta flexibilidad permite que el sistema se adapte a distintos escenarios de uso sin la necesidad de reconfigurar toda la arquitectura cada vez que cambia el entorno de destino.

1) ***Dockerfiles optimizados para CPU y GPU:***

   Permite la adaptación del sistema a entornos con o sin aceleración por GPU. Esto asegura una ejecución eficiente del procesamiento, optimizando el uso de recursos computacionales según la infraestructura disponible.

2) ***Docker Compose para entornos de desarrollo y producción.***

   Docker Compose facilita la definición y orquestación de los distintos servicios del sistema, permitiendo reproducir de forma consistente los entornos de desarrollo y producción mediante configuraciones centralizadas.

3) ***Configuración de redes y volúmenes.***

   Permite una comunicación controlada y segura entre los contenedores, mientras que los volúmenes garantizan la persistencia de datos como imágenes procesadas, resultados y registros del sistema, independientemente del ciclo de vida de los contenedores.

4) ***Implementación de un sistema dual de transferencia, que permite:***

a) Almacenamiento local de resultados:

Los resultados generados por el sistema se almacenan en directorios locales estructurados, asegurando disponibilidad inmediata y respaldo de la información procesada.

b) Transferencia automática vía HTTP/HTTPS:

Cuando se necesita integración con servicios externos, entra en juego la transferencia vía HTTP/HTTPS. Este mecanismo envía los resultados de forma automática mediante solicitudes HTTP o HTTPS, facilitando la comunicación con aplicaciones web u otras APIs sin intervención manual.

c) Transferencia automática vía FTP/FTPS/SFTP:

Para entornos más tradicionales o institucionales, el sistema también soporta **transferencia vía FTP, FTPS y SFTP**. Estos protocolos permiten enviar los resultados a servidores remotos y resultan especialmente útiles cuando se trabaja con infraestructuras que ya tienen este tipo de configuración establecida.

d) Configuración flexible para habilitar uno o ambos mecanismos simultáneamente:

Estos mecanismos no son excluyentes. El sistema puede configurarse para usar solo el almacenamiento local, solo la transferencia remota, o ambos al mismo tiempo, adaptándose sin fricciones a distintos requerimientos operativos.

3.17 Entorno de pruebas: Diferentes configuraciones de hardware

Para garantizar trazabilidad, reproducibilidad y consistencia con respecto a las métricas se realizaron pruebas catalogadas como experimentales, utilizando una laptop ASUS ROG Strix 16, que cuenta con un procesador Intel Core i9-13980HX de 24 núcleos, 16 GB de memoria RAM y con un sistema operativo Windows 11. Esta máquina brinda las capacidades necesarias para llevar a cabo la evaluación del rendimiento del sistema.

El microservicio desplegado en Docker, permitió un aislamiento estratégico de sus dependencias, manteniendo una configuración consistente del entorno y con condiciones similares a un ambiente de producción. Cabe recalcar que contenerizar permite hacer una

medición controlada acerca de la gestión de recursos evitando inferencias externas del entorno del host.

Con el fin de comparar el desempeño del sistema, se utilizaron los siguientes equipos y configuraciones:

- Segmentación haciendo uso de CPU.
- Equipo con tarjeta gráfica dedicada NVIDIA RTX 4060 (Tabla 4).
- Equipo con tarjeta gráfica dedicada NVIDIA RTX 3060 (Tabla 5).
- Equipo NVIDIA SPARK GB10 (Tabla 6).
- Equipo de pruebas HP de bajos recursos (Tabla 7).

Al probar en estas configuraciones, se pudo evaluar de manera concreta cómo la aceleración mediante hardware especializado afecta el rendimiento en tareas de segmentación profunda, especialmente en modelos tridimensionales de alta demanda computacional, como nnU-Net.

Las pruebas se realizaron con imágenes de resonancia magnética, que son volumétricas tridimensionales (3D), procesándolas de manera individual o en lotes. Esta estrategia permitió recrear condiciones similares a las de un entorno real, donde múltiples solicitudes pueden ejecutarse de forma concurrente o secuencial. Además, se mantuvo la trazabilidad de cada ejecución a través de los registros internos del sistema, lo que facilitó relacionar los tiempos de inferencia, el consumo de recursos y el comportamiento de los resultados.

Para complementar las pruebas, se realizaron pruebas en diferentes equipos, con distintas configuraciones de hardware con la finalidad de demostrar el rendimiento del microservicio.

En el caso de las pruebas realizadas con las computadoras en modo local, se las realizó con dos laptops, una funcionando como servidor y la otra para enviar las peticiones accediendo localmente a través de la IP. Con las configuraciones indicadas en la Tabla 4 y 5.

Tabla 5

Especificaciones del equipo de pruebas (ASUS NVIDIA GeForce RTX 4060)

Componente	Especificación
Equipo	ASUS ROG Strix G614JV
Procesador	13th Gen Intel® Core™ i9-13980HX (2.20 GHz)
Memoria RAM	16 GB DDR5 (15.6 GB utilizables)
Tarjeta gráfica principal	NVIDIA GeForce RTX 4060 Laptop GPU
Memoria VRAM dedicada	8 GB
GPU integrada	Intel UHD Graphics
Almacenamiento	954 GB SSD NVMe
Sistema Operativo	Windows 11 Home 64 bits
Versión del SO	25H2 (Build 26200.7840)
Versión DirectX	12 (FL 12.2)
Controlador GPU	32.0.15.9186 (20/01/2026)

Nota. Especificaciones obtenidas del sistema del equipo de pruebas utilizado para la experimentación

Tabla 6

Especificaciones del equipo de pruebas (ACER NVIDIA GeForce RTX 3060)

Componente	Especificación
Equipo	Acer Predator Helios 300
Procesador	12th Gen Intel® Core™ i7-12700H (2.70 GHz)
Memoria RAM	32 GB DDR5 (30.7 GB utilizables)
Tarjeta gráfica principal	NVIDIA GeForce RTX 3060 Laptop GPU
Memoria VRAM dedicada	6 GB
GPU integrada	Intel UHD Graphics
Almacenamiento	954 GB SSD NVMe
Sistema Operativo	Windows 11 Home 64 bits
Versión del SO	25H2 (Build 26200.7840)
Versión DirectX	12 (FL 12.2)
Controlador GPU	32.0.15.9579 (04/03/2026)

Nota. Especificaciones obtenidas del sistema del equipo de pruebas utilizado para la experimentación

Esta última laptop (Tabla 5) se la utilizó tanto en pruebas como servidor y como cliente de peticiones hacia la otra computadora.

Para el siguiente paso, y hacer pruebas en entornos de producción, se utilizó el microservicio ya desplegado en el equipo que se encuentra en la Unidad Académica de Posgrados de la Universidad Católica de Cuenca, en el laboratorio de investigación RobLab. El equipo “Spark”, cuenta con las siguientes especificaciones de hardware:

Tabla 7

Especificaciones del equipo de pruebas (SPARK)

Componente	Especificación
Servidor	Spark E4F9
Arquitectura	ARM64
CPU	ARM Cortex-X9 + Cortex-A725 (20 cores)
Memoria RAM	121 GB
Almacenamiento	3.7 TB (167 GB usados / 3.4 TB disponibles)
GPU	NVIDIA GB10
CUDA	13.0
Driver NVIDIA	580.126.09
Sistema Operativo	Ubuntu 24.04.4 LTS (Noble Numbat)
Kernel	6.17.0-1008-nvidia
Docker	29.1.3
Docker compose	v5.0.1
Entorno	Docker + GPU
Python	3.12.3
Modelo IA	nnU-Net (88M parámetros)
Dataset	BraTS 2023

Nota. Especificaciones obtenidas del sistema del equipo de pruebas utilizado para la experimentación

Además, se hicieron pruebas simultáneas, en una laptop de bajos recursos sin gráfica dedicada, con el objetivo de medir métricas, y observar el comportamiento del sistema bajo pruebas de estrés, ya que se enviaron lotes de imágenes desde dos computadoras al mismo tiempo. Las características de la computadora de bajos recursos son los siguientes:

Tabla 8

Especificaciones del equipo de pruebas (HP bajos recursos)

Componente	Especificación
Equipo	Hewlett-Packard
Procesador	Intel ® Core (TM) i7-470HQ @ 2.50GHz
Memoria RAM	8 GB DDR3
Tarjeta gráfica principal	Intel ® HD Graphics 4600
Memoria VRAM dedicada	128 Mb
GPU integrada	No disponible
Almacenamiento	932 GB disponibles
Sistema Operativo	Windows 10 Pro
Versión del SO	22H2
Versión DirectX	12 (FL 11.1)

Nota. Especificaciones obtenidas del sistema del equipo de pruebas utilizado para la experimentación

Para evaluar el comportamiento del microservicio en condiciones de hardware convencional, se realizaron varias pruebas ejecutando el modelo nnU-Net V2 en una PC

HP con procesador Intel Core i7-4710HQ a 2.50 GHz, 7.9 GB de RAM y sin GPU dedicada, contando únicamente con la gráfica integrada Intel HD Graphics 4600.

En cada prueba se enviaron solicitudes al endpoint POST /api/v1/predict/nnUNet utilizando imágenes del dataset BraTS 2023.

Para realizar estas pruebas, la máquina virtual se creó con la herramienta de VirtualBox, con la ISO original de Windows que se descargó a través de la página oficial de Microsoft, con la versión de 2022 22H2 estable. En cuanto a los recursos asignados de la máquina virtual, son los siguientes:

Tabla 9
Especificaciones del equipo de pruebas (VirtualBox)

Componente	Especificación
Equipo	Máquina Virtual (Oracle VirtualBox)
Procesador	Intel Core i9-13980HX (13th Gen) – 1 vCPU asignado (~2.4 GHz)
Memoria RAM	12 GB (12288 MB)
Tarjeta gráfica principal	VirtualBox Graphics Adapter (VBoxSVGA)
Memoria VRAM dedicada	128 MB
GPU integrada	No aplica en VM
Almacenamiento	50GB
Sistema Operativo	Windows 10 Pro 64 bits
Versión del SO	10.0.19045 (22H2)
Versión DirectX	DirectX 12

Nota. Especificaciones obtenidas del sistema del equipo de pruebas utilizado para la experimentación

3.18 Definición e implementación de herramientas para pruebas unitarias

Se desarrolló un infraestructura de benchmarking completa para evaluar el microservicio de segmentación. Las herramientas implementadas cubren dos modalidades de prueba.

Flujo A - Pruebas locales (desde el servidor SPARK)

Tabla 10
Pruebas locales (desde el servidor Spark)

Script	Funcion
benchmark_runner.py	Orquestador principal: envía imágenes y mide latencia, CPU, RAM y GPU por run
metrics_collector.py	Monitor de recursos del sistema, llamado internamente por el runner en cada inferencia
concurrency_test.py	Pruebas de concurrencia asíncrona con aiohttp (1, 2, 4, 8 usuarios simultáneos)
analyze.py	Análisis estadístico: media, std, IC95%, percentiles P50/P90/P95/P99, Mann-Whitney U
config.py	Configuración central del entorno experimental

Flujo B - Pruebas Externas desde PC hacia el SPARK

Tabla 11
Pruebas externas (desde PC hacia el Spark)

Script	Funcion
benchmark_runner_external.py	Runner con coordinación distribuida: mide latencia de red end-to-end
metrics_server.py	Servidor FastAPI en el Spark (puerto 9090): recolecta CPU, RAM y GPU por run_id
merge_results.py	merge_results.py Fusiona métricas cliente (PC) y servidor (Spark) por run_id

3.19 Protocolo experimental

Para la evaluación del sistema se han realizado varias ejecuciones del proceso de inferencia usando imágenes médicas que se tomaron del dataset BraTS 2023 utilizado en la investigación.

En cada ejecución se registraron métricas relacionadas con la latencia de inferencia, con el uso de memoria, con el uso de CPU/GPU y con la estabilidad del sistema.

Las pruebas realizadas se hicieron bajo un entorno de hardware configurado para el procesamiento de modelos de Deep Learning. Para cada escenario experimental se procesaron lotes de imágenes médicas para analizar cuál es su comportamiento en condiciones distintas de cargas. Posterior a la obtención de los resultados, estas fueron analizadas en el capítulo de resultados, en donde se comparan los valores registrados para cada métrica definida.

CAPÍTULO IV. RESULTADOS Y ANÁLISIS

4.1 Introducción

Este capítulo presenta y analiza de forma ordenada los resultados obtenidos después de implementar y ejecutar el microservicio de segmentación de imágenes médicas, el cual fue descrito en el Capítulo III, donde se explicó el diseño de la arquitectura, las decisiones metodológicas adoptadas y las tecnologías utilizadas durante el desarrollo. A partir de ello, esta sección se enfoca en evaluar, mediante pruebas en un entorno controlado, cómo se comporta el sistema en la práctica y cuáles son los resultados que produce.

El análisis se basa en las métricas cuantitativas valorando su desempeño en condiciones de operación, con estos datos se observa eficiencia computacional, estabilidad y capacidad de procesamiento de información. Durante la segmentación se revisa la latencia, consumo de recursos como CPU, GPU y memoria, y tasa de errores. El comportamiento general de las salidas que son generadas por los modelos de segmentación también se considera para obtener una retroalimentación del sistema en cuanto a funcionamiento técnico.

Cabe aclarar que los resultados no corresponden a una validación clínica de los modelos ni a una evaluación diagnóstica del sistema. El análisis se limita exclusivamente al desempeño técnico del microservicio, entendido como una infraestructura computacional diseñada para ejecutar modelos de segmentación previamente entrenados.

4.2 Resultados técnicos del sistema

4.2.1 Latencia de inferencia

La latencia de inferencia corresponde al tiempo que transcurre desde que el microservicio recibe una imagen hasta que se genera por completo el resultado segmentado. Este proceso no solo incluye la ejecución del modelo, sino también las etapas previas de preprocesamiento y las posteriores de postprocesamiento, que forman parte del flujo necesario para obtener el resultado final.

En las pruebas experimentales se evaluó el desempeño de dos modelos de segmentación bajo las configuraciones de CPU y GPU en entorno local y en entorno distribuido. Para medir la latencia de inferencia, se procesó un conjunto de imágenes de resonancia magnética en formato NIfTI, correspondientes a volúmenes tridimensionales (3D), con un tamaño promedio de 2.53 MB

En el caso de nnU-Net, se realizaron diez ejecuciones consecutivas del modelo nnU-Net V2 (PyTorch) con aceleración mediante GPU dedicada NVIDIA GeForce RTX 4060 (8 GB VRAM), empleando las mismas imágenes NIfTI del dataset BraTS 2023. Los resultados se presentan en la Tabla 9.

Al habilitar la aceleración por GPU, utilizando una NVIDIA RTX 4060, la latencia se redujo de forma notable. Esta mejora representa una reducción considerable en comparación con la ejecución en CPU, lo que evidencia que el modelo depende en gran medida de operaciones paralelizables y de alto costo computacional, especialmente aquellas relacionadas con el cálculo matricial. El uso de la GPU permitió aprovechar de manera eficiente el paralelismo masivo que requiere la arquitectura tridimensional.

Tabla 12
Resultados de inferencia en GPU

Equipo de prueba: ASUS ROG Strix G16 (G614JV)

Prueba	Tiempo (s)	Uso GPU (%)	VRAM (GB)	Tamaño Imagen (MB
1	16.481	85	0.4	2,52 MB
2	17.787	92	0.4	2,37 MB
3	17.336	90	0.4	2,70 MB
4	11.563	90	0.4	2,52 MB
5	17.617	97	2.1	2,37 MB
6	17.606	43	0.6	2,70 MB

7	11.714	29	0.6	2,52 MB
8	17.521	62	0.6	2,37 MB
9	18.301	43	0.4	2,70 MB
10	10.624	21	0.6	2,52 MB

Nota. Se realizaron 10 ejecuciones consecutivas del modelo utilizando GPU

Por otro lado, el modelo desarrollado en TensorFlow registró tiempos de procesamiento mucho más bajos en ambas configuraciones. Se realizaron diez ejecuciones consecutivas del modelo UNet 3D MultiScale (TensorFlow) con la GPU desactivada, utilizando imágenes NIfTI del dataset BraTS 2023 con un tamaño promedio de entre 2.37 MB y 2.70 MB. Los resultados se presentan en la Tabla 10.

Se registraron diez ejecuciones consecutivas del microservicio en modo CPU. La primera ejecución registró un tiempo de 2.36 segundos, valor superior al resto debido a la carga inicial del modelo en memoria. A partir de la segunda ejecución, los tiempos se estabilizaron en un rango de entre 1.53 y 1.66 segundos, con un tiempo promedio final de 1.58 segundos por imagen. El consumo de CPU se mantuvo en un rango de entre 4% y 7%, mientras que el uso de memoria RAM permaneció estable entre 10.7 GB y 10.9 GB durante todas las ejecuciones. La ausencia de incrementos progresivos en el consumo de recursos indica que no se produjeron fugas de memoria, lo que evidencia estabilidad operativa del sistema.

Tabla 13
Resultados de inferencia en CPU

Equipo de prueba: ASUS ROG Strix G16

Prueba	Tiempo (s)	Uso CPU (%)	RAM (GB)	Tamaño Imagen (MB
1	2.360	~5%	10.3	2,52 MB
2	1.580	~4%	10.3	2,37 MB
3	1.568	~4%	10.3	2,70 MB
4	1.568	~5%	10.6	2,52 MB
5	1.543	~7%	10.8	2,37 MB
6	1.533	~5%	10.8	2,70 MB
7	1.598	~6%	10.8	2,52 MB
8	1.594	~5%	10.7	2,37 MB
9	1.567	~4%	10.8	2,70 MB
10	1.665	~4%	10.9	2,52 MB

Nota. Estos son los resultados tras 10 ejecuciones consecutivas con GPU desactivada.

Promedio tiempo CPU: 1.58 segundos

Uso promedio CPU: 5.1 %

Consumo promedio RAM: 10.77 GB

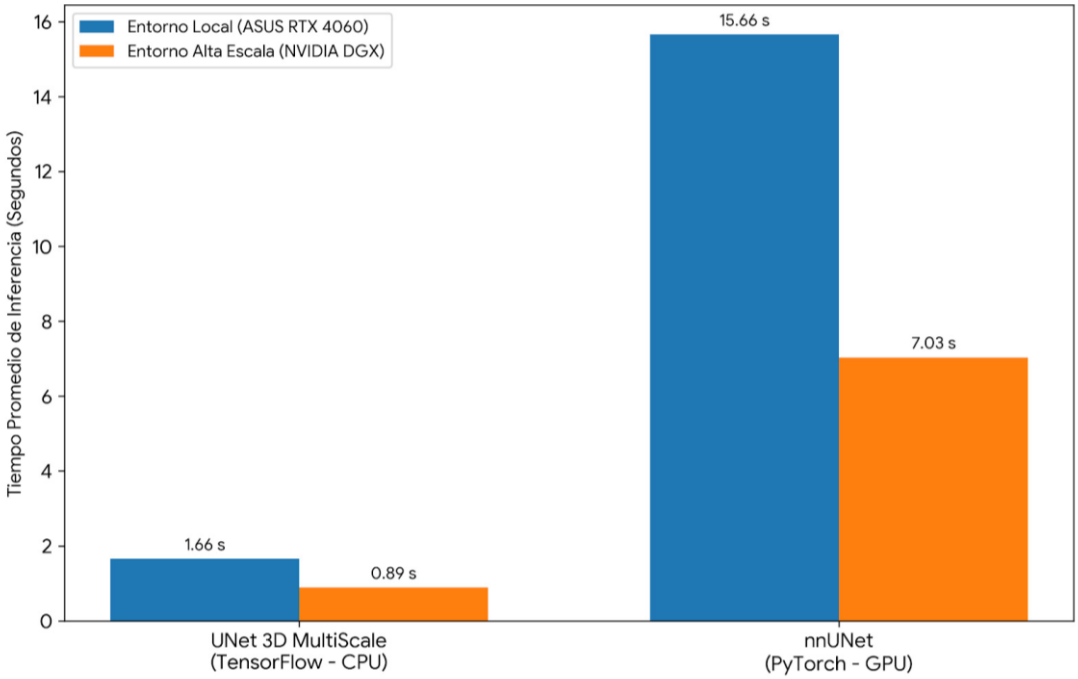
En cuanto a la estabilidad del sistema, la tasa de errores fue baja en ambas configuraciones, presentando únicamente casos aislados sin impacto significativo en la ejecución general del microservicio.

Como se observa en la figura 4, el tiempo de ejecución varía de manera significativa en función de la complejidad arquitectónica del modelo empleado. El despliegue del modelo nnUNet en la GPU local toma en promedio 15.66 segundos, sin embargo, su ejecución

en un entorno distribuido de alto rendimiento (NVIDIA DGX) reduce este tiempo en más de un 55% (7.03 s). El modelo ligero UNet, por su parte, reportó latencias inferiores a los 2 segundos incluso sin aceleración por GPU local.

Figura 4

Comparación de latencia de inferencia por modelo y entorno

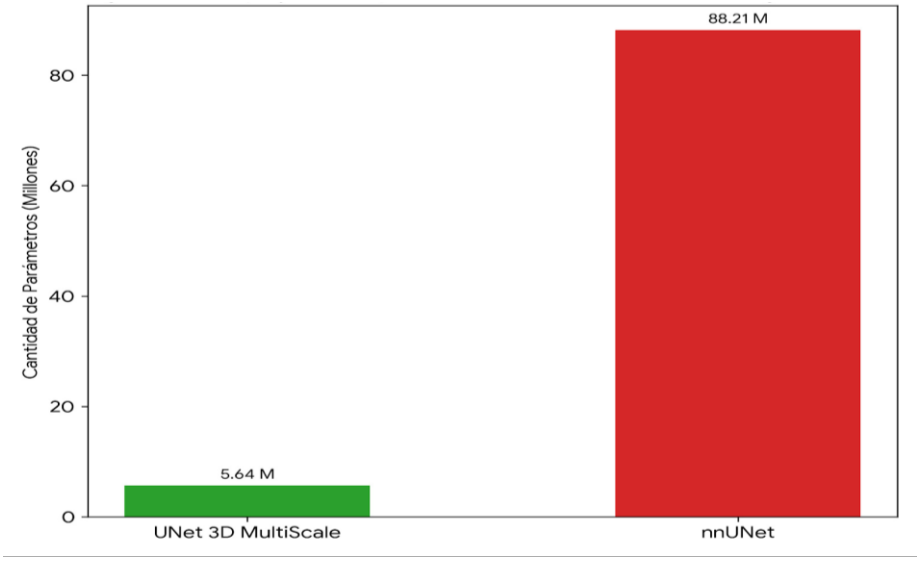


Nota. Elaboración propia.

La diferencia sustancial en los tiempos de inferencia y uso de VRAM entre los dos modelos está directamente correlacionada con su profundidad matemática. Mientras que la arquitectura base (UNet 3D) cuenta con 5.64 millones de parámetros entrenables, nnUNet opera con más de 88 millones de parámetros (Figura 5), requiriendo 15 veces más carga de operaciones de tensor y justificando el uso intensivo de la GPU (hasta 96% de uso en la NVIDIA DGX).

Figura 5
Complejidad arquitectónica de los modelos de segmentación integrados.

Nota. Elaboración propia.



En conclusión, los resultados confirman que la aceleración por hardware mediante GPU proporciona una mejora considerable en tiempos de inferencia, especialmente en modelos de mayor complejidad computacional, optimizando así el rendimiento global del microservicio.

4.2.2 Uso de memoria y GPU

El análisis llevado a cabo con respecto al consumo de recursos tanto de memoria como de gráficos, ha permitido realizar una evaluación en cuanto a la estabilidad del sistema en ejecuciones continuas y prolongadas. Una gestión incorrecta de la memoria puede generar una disminución del rendimiento, inclusive provocando fallos inesperados.

Durante la carga de los modelos, se observó un incremento en el consumo de la memoria RAM y en la GPU, esto es normal debido a que en sistemas que están basados en redes neuronales, se requiere un consumo de recursos elevado. En la inferencia (segmentación), los modelos necesitan mantener en memoria los parámetros como los tensores que generan en el procesamiento.

Después de la segmentación, al finalizar la carga inicial, se pudo observar estabilidad en los recursos como de memoria y aceleración por hardware, en dónde no demostró picos ni incrementos acumulativos fuera de lo esperado. Los recursos temporales indican que

se liberan correctamente, dando como finalidad un ciclo de vida correcto y bien gestionado en lo que respecta del pipeline de procesamiento.

En cuanto al uso específico de la GPU, una NVIDIA RTX 4060, se observó un consumo moderado durante la fase de inferencia, el cual varió según el modelo utilizado y el tamaño del lote procesado. En el caso de nnU-Net, la ocupación del dispositivo fue mayor, lo que se explica por el uso de convoluciones tridimensionales, que requieren más recursos. Por el contrario, el modelo implementado en TensorFlow presentó un consumo menor, lo que refleja una menor demanda computacional durante su ejecución.

Ahora en cuanto a la otra laptop (Acer) que se utilizó para las pruebas de enviar lotes de imágenes, utilizando una tarjeta gráfica Nvidia RTX 3060 con 6 GB de memoria, se observó un consumo moderado durante el proceso de segmentación, utilizando el modelo de nnU-Net v2.

Es importante señalar que, en ninguno de los escenarios evaluados, se alcanzó un nivel crítico de saturación de la memoria gráfica ni se presentaron errores relacionados con la falta de recursos. El sistema se mantuvo dentro de límites operativos normales durante todas las pruebas. Esto demuestra que el microservicio puede funcionar de forma estable en el entorno definido, manteniendo un balance adecuado entre la carga computacional y los recursos de hardware disponibles.

4.2.3 Tasa de errores

La tasa de errores es un indicador necesario para conocer la robustez del software mostrando la frecuencia en la que se presentan fallos, entre ellos destacan: fallos durante carga de imágenes, ejecuciones del modelo o generación de resultados de segmentaciones. En el microservicio se consideran errores propios, es decir, por entradas inválidas. Como se muestra en la tabla 11, los resultados obtenidos durante las pruebas evidencian una tasa de error del 0%, evidenciando estabilidad y robustez del sistema bajo condiciones controladas.

Tabla 14

Contabilización de errores

Modelo	Total pruebas	Ejecuciones exitosas	Errores	Tasa de error
UNet 3D MultiScale(CPU)	10	10	0	0%
nnUNet (GPU)	10	10	0	0%

Los incidentes detectados estuvieron relacionados principalmente con archivos corruptos, formatos no compatibles o parámetros de ejecución incorrectos, los cuales se introdujeron de forma intencional para verificar el funcionamiento de los mecanismos de validación.

El sistema de segmentación respondió de forma adecuada debido a los mecanismos de control de excepciones que se implementaron en la capa de servicio, evitando caídas del contenedor o interrupciones en su funcionamiento. Para cada fallo, se generaron logs y mensajes de error claros, identificando la causa, sin afectar la estabilidad del microservicio.

Sin embargo, cuando los parámetros si se cumplían con los criterios establecidos el sistema funcionó correctamente, sin interrupciones, incluso en el endpoint de batch (segmentación por lotes). Confirmando que estrategias de validación de datos, conjuntamente con el manejo de errores contribuyen efectivamente a tener estabilidad operativa del sistema.

4.2.4 Distribución de clases predichas

Analizar la distribución de clases predichas tiene como objetivo la deteccion de posibles anomalías en la salida del modelo por ejemplo, la generación de predicciones pueden indicar fallos en la segmentación, o que se colapse hacia una sola clase.

Las imágenes utilizadas son de BraTS, la variabilidad de los datos de segmentaciones de tumores cerebrales tuvo coherencia con la distribución observada. Se generaron máscaras,

para distinguir las regiones tumorales del tejido sano, sin pruebas de sesgos marcados ni tampoco patrones de anomalías del modelo.

Cabe recalcar que este análisis no contiene una evaluación formal con métricas como Dice Score o IoU. La investigación presente tuvo como propósito verificar la consistencia técnica durante la segmentación, garantizando que las predicciones mantengan una estructura lógica que esté acorde al problema identificado.

El funcionamiento del pipeline de preprocesamiento, normalización y procesamiento, es correcto en el entorno de ejecución establecido ya que no se presentan colapsos sistemáticos.

4.2.5 Resultados en entornos de producción/local del equipo SPARK

Los resultados obtenidos a partir de la ejecución de pruebas unitarias y masivas sobre el microservicio de segmentación de tumores cerebrales, en un entorno simulado de producción local y distribuido (externo), evidencian un desempeño altamente estable, consistente y acorde a los requerimientos de sistemas clínicos basados en IA. La evaluación secuencial realizada sobre el dataset de BraTS 2023, con 30 ejecuciones válidas arrojó un promedio en la latencia de 7.899 ± 0.506 segundos (Tabla 12), con un intervalo de confianza del 95% comprendido entre 7.710 y 8.088 segundos. Estos valores reflejan una baja variabilidad en los tiempos de respuesta, lo cual constituye un indicador clave de confiabilidad operativa. Asimismo, el percentil 95 de 8.044 segundos confirma que en escenarios cercanos al peor caso, el sistema mantiene un comportamiento predecible. Es importante destacar que no se han registrado errores durante las ejecuciones (error rate = 0.0%), lo que valida la robustez del microservicio y la correcta integración del modelo nnU-Net V2 en el entorno de inferencia.

Tabla 15
Resultados – Rendimiento secuencial (30 imágenes BraTS 2023)

Métrica	Local (Spark)	Externo (PC → SPARK)
Latencia media	7.899s ± 0.506s	8.027s ± 0.613s
IC95%	[7.710 – 8.088]	[7.798 – 8.256]
Latencia mínima	6.02s	6.05s
Latencia máxima	8.06s	9.09s
P95	8.044s	9.052s
Error rate	0.0%	0.0%
GPU Utilización media	—	~47%
RAM usada	~10.8 – 11.7 GiB	~10.7 – 11.5 GiB
Overhead de red	—	~128ms (~1.6%)
Runs válidos	30/30	30/30

Impacto de la infraestructura y latencia de red

El análisis comparativo entre el escenario local (ejecución en el servidor) y escenario externo (desde una pc que actúa como cliente en red para hacer la petición) permitió determinar que el overhead de red es mínimo y no representa un factor limitante en el rendimiento del sistema.

En términos técnicos la latencia promedio en entorno externo fue de 8.027 ± 0.613 segundos, apenas superior al entorno local, con una diferencia aproximada de 128 milisegundos. Este resultado demuestra que la arquitectura distribuida implementada es eficiente y que la comunicación en red no introduce degradaciones significativas. En consecuencia, se concluye que el principal cuello de botella del sistema reside en el proceso de inferencia del modelo, lo cual es esperable dada la complejidad computacional del mismo.

Análisis de escalabilidad y comportamiento bajo concurrencia

En relación con la capacidad de escalamiento del microservicio, las pruebas de concurrencia evidencian un comportamiento consistente basado en un modelo de procesamiento secuencial controlado por GPU. El throughput se mantuvo constante en

aproximadamente 0.13 solicitudes por segundo, independientemente del número de usuarios. Este fenómeno se explica por la naturaleza del procesamiento en la GPU, donde las inferencias son gestionadas en una cola de tipo FIFO (First-In, First-Out), permitiendo la ejecución de una tarea a la vez. A pesar del incremento en la latencia percibida por solicitud bajo mayores niveles de concurrencia (alcanzando valores de hasta 102.368 segundos en el P95 para 8 usuarios), no se registraron errores en ningún escenario, lo que confirma la estabilidad del sistema bajo carga. Este comportamiento, es particularmente relevante en contextos clínicos, donde la consistencia y la ausencia de fallos son prioritarios frente a la latencia en situaciones de alta demanda.

Tabla 16
Resultados – Escalabilidad bajo carga (Concurrencia)

Usuarios	Throughput Local	Throughput Externo	P50 Local	P50 Externo	P95 Local	P95 Externo	Errores
1	0.13 req/s	0.12 req/s	8.025s	8.042s	8.041s	8.051s	0.0%
2	0.13 req/s	0.12 req/s	16.043s	16.062s	24.071s	18.065s	0.0%
4	0.13 req/s	0.12 req/s	24.086s	32.086s	78.282s	41.118s	0.0%
8	0.13 req/s	0.12 req/s	56.153s	57.160s	102.368s	88.233s	0.0%

Validación del entorno experimental y reproducibilidad

Un aspecto fundamental de los resultados radica en la rigurosidad del entorno experimental implementado. La utilización de un protocolo de calentamiento que descarta ejecuciones iniciales afectadas por el fenómeno de cold start, garantiza la reproducibilidad de los experimentos y la validez estadística de los resultados. Adicionalmente, el monitoreo en tiempo real de recursos (CPU, RAM y GPU), el aislamiento de cada inferencia mediante indicadores únicos y la inclusión de mecanismos de encriptación en las comunicaciones, permiten replicar condiciones cercanas a un entorno real clínico. La estabilidad en el uso de memoria y la utilización promedio de la GPU (~47%) evidencian una adecuada gestión de los recursos, evitando saturaciones o comportamientos anómalos durante la ejecución.

Los resultados de las pruebas mencionadas en el anterior capítulo, en el entorno de pruebas, en donde se realizaron 10 ejecuciones, demuestran que el microservicio respondió correctamente con estado HTTP 200 OK, detectando las 4 clases de segmentación esperadas. Los tiempos registrados rondaron consistentemente los 12 minutos por inferencia, un valor que se mantuvo estable entre prueba y prueba. Durante las ejecuciones se observó que el procesador trabajó alrededor del 77% de su capacidad y la RAM alcanzó el 86% de uso (6.8 de 7.9 GiB), lo que indica que el equipo operaba cerca de su límite solo para sostener una inferencia a la vez. El modelo nnU-Net V2 no es viable en entornos únicamente CPU debido a tiempos de inferencia elevados que superan los 300 segundos por imagen.

Para finalizar, también se realizaron pruebas con una máquina virtual para simular un equipo básico que no cuenta con GPU, y con recursos limitados, con el objetivo de tener una retroalimentación de cómo se desempeña el microservicio en este tipo de entornos con recursos computacionales básicos.

En este contexto, el microservicio desde una máquina virtual con Windows 10, se envió la imagen BraTS-GLI-00002-000-t1c.nii.gz al endpoint POST /api/v1/predict/nnUNet. La inferencia se completó correctamente con HTTP 200 OK, detectando las 4 clases de segmentación, pero el tiempo fue de 30 minutos y 5 segundos el más alto de todos los entornos evaluados. La combinación de la sobrecarga de virtualización, el uso de un modelo de segmentación pesado y el uso solamente de CPU explica buena parte de esa diferencia.

Comparado con los 7.9 segundos del servidor Spark con GPU NVIDIA GB10, este entorno resultó aproximadamente 226 veces más lento, lo que confirma que la virtualización sin GPU dedicada no es viable para un contexto clínico real.

Conclusión de desempeño y validación del sistema

En síntesis, los resultados obtenidos demuestran de manera concluyente que el microservicio de segmentación basado en el modelo de nnU-Net V2, desplegado sobre la infraestructura SPARK con GPU NVIDIA GB10, cumple con los criterios de rendimiento, estabilidad y confiabilidad exigidos en aplicaciones de diagnóstico asistido

por IA. La combinación de una latencia predecible, una tasa de error nula, un impacto mínimo de la red y un comportamiento estable bajo concurrencia, respalda la viabilidad del sistema para su integración en entornos clínicos reales. Por tanto, se valida no solo la correcta implementación técnica del microservicio, sino que también la metodología experimental aplicada, consolidando los resultados como un aporte sustentado dentro del marco del proyecto de investigación.

4.3 Discusión de resultados

En cuanto a los resultados obtenidos, la arquitectura dividida en módulos, y el pipeline desglosado en preprocesamiento, segmentación y postprocesamiento, han permitido alcanzar una estabilidad en el ámbito operativo, gestionando eficientemente cada recurso (CPU/GPU). Estos resultados están alineados a las investigaciones por parte de Magadza y Viriri, que han reportado que la optimización del modelo nnU-Net reduce considerablemente los tiempos de respuesta en la segmentación, manteniendo la inferencia de tumores cerebrales con el dataset de BraTS 2020. (Magadza & Viriri, 2023). Debido a que su estudio está centrado únicamente en la eficiencia del modelo, el presente análisis va más allá ya que incluye métricas tanto operativas como de escalabilidad, además brinda un enfoque más completo dentro de sistemas de producción en un entorno contenerizado.

Los resultados en cuanto a tiempos de latencia y throughput hacen énfasis en el uso de la aceleración por hardware (GPU), ya que reduce su respuesta en modelos 3D complejos como lo es nnU-Net, ahora en modelos menos complejos, por ejemplo, los recursos se mantienen más estables ofreciendo un mejor desempeño, ya que un factor importante de este modelo es la portabilidad en entornos donde los recursos son un poco más limitados, haciendo uso de CPU (gráficos integrados). Éstas observaciones concuerdan con investigaciones sistemáticas realizadas por Hamza y Damaševičius, demostrando que los modelos de deep learning necesitan de recursos de GPU para obtener un desempeño alto y eficiente, reduciendo los tiempos de throughput. Es importante cumplir con las

estrategias como lo es el batch processing, con el objetivo de alcanzar una mejora en la escalabilidad operativa (Hamza & Damaševičius, 2025).

Desde el punto de vista de robustez y manejo de errores, se ha controlado que el microservicio desarrollado integre métricas de seguridad al momento de la ejecución, como lo es el cifrado de datos que se ha incluido en el sistema. Para comparar estos resultados, hay estudios que han demostrado que aunque se alcancen valores altos de precisión en segmentación, no hay investigaciones que contengan una evaluación con respecto a la estabilidad operativa, o incluyan métricas de seguridad ya en un ambiente de producción, es por esto que la presente investigación constituye un aporte diferencial (Lu et al., 2025). Incluso, tener la capacidad de hacer ejecuciones de inferencia con diferentes configuraciones de recursos manteniendo el flujo de datos ya destaca la importancia de portabilidad y escalabilidad en ambientes clínicos reales.

La distribución de clases predichas y las máscaras generadas confirman que el pipeline ha sido capaz de mantener coherencia y constancia técnica cuando se hace la segmentación. Para complementar estos resultados, están los hallazgos de Lu et al., en donde se reportan cuales son las técnicas que fusionan canales y preprocesamiento, con la finalidad de incrementar la calidad de segmentación, aunque el estudio no aborda métricas de eficiencia operativa ni tampoco consideraciones para un despliegue real (Lu et al., 2025).

Para concluir, cada uno de los resultados operativos en cuestión (latencia, throughput, uso de recursos), han demostrado que hacer uso de contenedores permite controlar la ejecución de modelos de manera precisa, inclusive con distinto hardware. Esta propuesta se diferencia de otras en donde únicamente se centran en métricas segmentación, ya que esta incluye una evaluación técnica en cuanto al desempeño, al uso de recursos, y estabilidad del sistema. Además, como se mencionó anteriormente, incorpora mecanismos de seguridad, como lo es el cifrado de datos y control en entradas, que protege información y datos sensibles. Estos resultados indican que el microservicio se

ajusta a los estándares de segmentación para tumores cerebrales, pero también, se toma en cuenta aspectos claves como lo son la portabilidad, escalabilidad y seguridad. Se posiciona como una opción viable para una implementación en entornos clínicos reales.

4.4 Análisis cuantitativo complementario

Se realizó una estimación acerca de la capacidad operativa que tiene el sistema, considerando que la cantidad de imágenes procesadas por el tiempo de una hora pueden variar de acuerdo a los recursos de hardware que estén configurados. De esta forma, se puede dimensionar el rendimiento del sistema en uso continuo.

Con un promedio en latencia entre 20 a 25 segundos por imágen usando el modelo pesado (nnUNet) con GPU, el sistema podría procesar aproximadamente entre 144 a 180 por una hora de inferencia de corrido. Sin embargo, usando CPU, la latencia aumenta a un promedio de 180 a 240 segundo por imágen usando el mismo modelo, su capacidad es disminuida de forma considerable, ya que puede procesar entre 15 a 20 imágenes por hora.

Usando el modelo liviano (UNet 3D Multiescala), al ejecutarse con GPU, la latencia promedio se sitúa entre 8 a 12 segundos, alcanzando un estimado de 300 a 450 imágenes por hora, representando una gran mejora en el rendimiento del sistema.

A pesar de que se trata de valores estimados, permiten comprender el comportamiento del sistema en entornos hospitalarios donde el volumen de estudios suele ser elevado. Usar GPU es un factor clave ya que asegura que el sistema opere eficientemente con tiempos de respuesta precisos.

Procesar imágenes por lote (batch), mejora la eficiencia general, ya que no es necesario cargar el modelo múltiples veces, aprovechando los recursos disponibles de mejor manera. Este comportamiento puede ser escalable efectivamente si se despliega en infraestructuras con mejores capacidades de recursos a nivel de cómputo.

4.5 Limitaciones del estudio

El presente estudio no lleva a cabo una evaluación clínica formal sobre la segmentación de las imágenes médicas, empleando métricas estándar como por ejemplo coeficiente de Dice, sensibilidad, especificidad e Intersection over Union (IoU), determinando así cual ha sido su diagnóstico real del modelo en ambientes clínicos.

La escalabilidad horizontal, mediante replicación de contenedores orquestada con Kubernetes, no fue implementada en el presente estudio. Sin embargo, la arquitectura modular y contenerizada desarrollada constituye una base técnica adecuada para habilitar dicha escalabilidad en una fase posterior, lo cual es consistente con los principios de diseño cloud-native adoptados.

4.6 Implicaciones técnicas

Posterior a los resultados obtenidos, se han identificado algunas implicaciones para un futuro desarrollo. Desde el punto de vista de la infraestructura, se pudo observar una notable disminución en la latencia al usar aceleración por hardware (GPU), indicando que el sistema rinde mejor en ambientes donde se cuenta con recursos favorables, es decir, múltiples GPU. Además, integrar el sistema con plataformas como Kubernetes, permite una escalabilidad horizontal asignando recursos dinámicamente con respecto a su demanda, resultando útil en escenarios donde la carga varíe.

En cuanto a la arquitectura realizada por módulos ha facilitado la integración de más modelos sin interferir en la capa de presentación o de servicio, permitiendo la integración de arquitecturas y modelos con mejor optimización con técnicas de cuantización y pruning para reducir aún más los recursos de hardware.
En conjunto, estos resultados nos demuestran que el microservicio desarrollado es técnicamente viable y, al mismo tiempo, sientan una base para su crecimiento. Esto permite que, en el futuro, pueda convertirse en un sistema más robusto, escalable y con potencial para integrarse en entornos clínicos reales.

CAPÍTULO V. CONCLUSIONES

El objetivo de esta investigación es el diseño, implementación y evaluación técnica de un microservicio que se orienta a segmentar imágenes médicas automáticamente con modelos de deep learning. Con los resultados obtenidos se destaca que las decisiones metodológicas que se integraron para desarrollar el sistema fueron las adecuadas y coherentes con los objetivos planteados.

Como primer punto, los resultados de la arquitectura en capas propuesta, permitió estructurar las responsabilidades del software, haciendo la separación entre las diferentes capas: presentación, servicio e inferencia. Permitiendo un sistema mantenible, escalable y con trazabilidad del microservicio. Además, permite la integración de modelos entrenados en varios frameworks como TensorFlow y PyTorch.

Como segundo punto a destacar, los resultados obtenidos han demostrado un desempeño técnico bajo diferentes configuraciones de hardware dentro del sistema. Se evaluó la segmentación utilizando CPU y GPU dando como resultado menores tiempos en latencia de inferencia, brindando soporte para procesamientos con diferentes tipos de recursos.

Durante ejecuciones continuas, el microservicio se mantuvo estable y demostró robustez, además de controlar los recursos, memoria y GPU que garantiza estabilidad. También, se observó que se evitaron fallos críticos al implementar validaciones de entrada, asegurando disponibilidad. El sistema demostró que no solamente es funcional sino que también es confiable dentro del entorno evaluado.

El alcance de la presente investigación fue la evaluación técnica al sistema como una infraestructura computacional para ejecutar modelos entrenados. Cabe aclarar que no se realizó ninguna validación clínica formal ni un análisis del desempeño diagnóstico del modelo. Sin embargo, el uso de datos del dataset de Brain Tumor Segmentation Challenge (BraTS), aseguró coherencia estructural en las segmentaciones generadas, alineados con estándares aceptados en investigación de segmentación biomédica.

Se concluye que el sistema es viable, cumpliendo con los objetivos planteados y constituye una base para continuar con investigaciones futuras. El microservicio implementado es capaz de integrar modelos de segmentación dentro de una arquitectura contenerizada, es escalable y está orientado a servicios, manteniendo un cumplimiento de métricas críticas de desempeño.

5.1 Aportes del trabajo

Los resultados obtenidos han permitido hacer avances importantes al ejecutar modelos de deep learning en sistemas de ejecución desplegables y en ingeniería de software.

Las aportaciones más notorias son:

- Diseño y ejecución de una arquitectura modular de sistema para el procesamiento de modelos de segmentación de imágenes médicas.
- Implementación de los diferentes frameworks de aprendizaje profundo en un único microservicio.
- Estudio comparativo técnico de los resultados en CPU, GPU y en diferentes escenarios.
- Integración de todo el procesamiento por lotes como un método de mejora del rendimiento.
- Aplicación de métricas operativas para el estudio del rendimiento.

5.2 Trabajos futuros

Después de los resultados obtenidos, una propuesta para una posible investigación a futuro, sería implementar una evaluación formal de desempeño clínico mediante métricas estandarizadas, verificando el funcionamiento del propio modelo en un entorno clínico. Y, observar cómo funciona el microservicio ocupando varias sesiones en paralelo, mediante mecanismos de orquestación como Kubernetes con el fin de estudiar la capacidad del sistema y el cómo la carga se distribuye.

Para hacer el proyecto escalable, se pueden realizar varias investigaciones con el fin de mejorar los modelos, es decir, optimizar tiempos de respuesta, gestionar mejor los

recursos, etc., con la implementación de técnicas de aprendizaje automático como la cuantización, por ejemplo. Pero estas mejoras no deben afectar el funcionamiento del modelo.

Otro propuesta para continuar con la investigación está dirigida a la utilización de microservicios ya en sistemas hospitalarios, como los sistemas PAC o las historias clínicas electrónicas. Ya que se podría investigar cómo los microservicios se pueden adaptar a los flujos de trabajo y mejorar la atención del paciente. Así mismo, profundizar en la evaluación de seguridad y cumplimiento normativo en estos mismos entornos clínicos.

REFERENCIAS

Bathelt, F., Lorenz, S., Weidner, J., Sedlmayr, M., & Reinecke, I. (2025). Application of Modular Architectures in the Medical Domain—A Scoping Review. *Journal of Medical Systems*, *49*(1), 27. https://doi.org/10.1007/s10916-025-02158-3

Brundage, D., Rosenthal, J., Carelli, R., Rand, S., Umeton, R., Loda, M., & Marchionni, L. (2022). *Whole Slide Image to DICOM Conversion as Event-Driven Cloud Infrastructure* (arXiv:2203.13888). arXiv. https://doi.org/10.48550/arXiv.2203.13888

Faseeha, U., Jamil Syed, H., Samad, F., Zehra, S., & Ahmed, H. (2025). Observability in Microservices: An In-Depth Exploration of Frameworks, Challenges, and Deployment Paradigms. *IEEE Access*, *13*, 72011–72039. https://doi.org/10.1109/ACCESS.2025.3562125

Garg, S., Pundir, P., Rathee, G., Gupta, P. K., Garg, S., & Ahlawat, S. (2022). *On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps* (arXiv:2202.03541). arXiv. https://doi.org/10.48550/arXiv.2202.03541

Gomes, F., Rego, P., & Trinta, F. (2025). A systematic mapping study on observability of microservices-based applications: Fundamentals, classifications, and challenges. *Computing*, *107*(9), 183. https://doi.org/10.1007/s00607-025-01540-w

Gupta, V., Erdal, B., Ramirez, C., Floca, R., Genereaux, B., Bryson, S., Bridge, C., Kleesiek, J., Nensa, F., Braren, R., Younis, K., Penzkofer, T., Bucher, A. M., Qin, M. M., Bae, G., Lee, H., Cardoso, M. J., Ourselin, S., Kerfoot, E., … Shuaib,

H. (2024). Current State of Community-Driven Radiological AI Deployment in Medical Imaging. *JMIR AI*, *3*(1), e55833. https://doi.org/10.2196/55833

Hamza, A., & Damaševičius, R. (2025). Deep Learning for Brain Tumor Segmentation and Classification: A Systematic Review of Methods and Trends. *Computers, Materials & Continua*, *86*(1), 1–41. https://doi.org/10.32604/cmc.2025.069721

Harshith, M., Ansari, Z. A., Fatima, S., Siddiqui, S., Swarna, S., Reddy, D. R. N., & Mohsin, S. W. (2026). Federated microservices architecture with blockchain for privacy-preserving and scalable healthcare analytics. *Scientific Reports*, *16*(1), 9023. https://doi.org/10.1038/s41598-026-39837-1

Hewage, N., & Meedeniya, D. (2022). *Machine Learning Operations: A Survey on MLOps Tool Support*. https://doi.org/10.48550/arXiv.2202.10169

Huang, J., & Liu, Y. (2025a). MLXOps4Medic: A Service Framework for Machine Learning and Explainability Operations in Medical Imaging AI Development. *IEEE Access*, *13*, 158149–158169. https://doi.org/10.1109/ACCESS.2025.3606838

Huang, J., & Liu, Y. (2025b). MLXOps4Medic: A Service Framework for Machine Learning and Explainability Operations in Medical Imaging AI Development. *IEEE Access*, *13*, 158149–158169. https://doi.org/10.1109/ACCESS.2025.3606838

Li, B., Peng, X., Xiang, Q., Wang, H., Xie, T., Sun, J., & Liu, X. (2022). Enjoy your observability: An industrial survey of microservice tracing and analysis. *Empirical Software Engineering*, *27*(1), 25. https://doi.org/10.1007/s10664-021-10063-9

Lu, N.-H., Huang, Y.-H., Liu, K.-Y., & Chen, T.-B. (2025). Deep learning-driven brain tumor classification and segmentation using non-contrast MRI. *Scientific Reports*, *15*(1), 27831. https://doi.org/10.1038/s41598-025-13591-2

Magadza, T., & Viriri, S. (2023). Efficient nnU-Net for Brain Tumor Segmentation. *IEEE Access*, *11*, 126386–126397. https://doi.org/10.1109/ACCESS.2023.3329517

Nopour, R. (2026). Using FHIR for data sharing: A scoping review of challenges and facilitators in healthcare settings. *International Journal of Medical Informatics*, *205*, 106128. https://doi.org/10.1016/j.ijmedinf.2025.106128

Panchal, D., Baran, I., Musgrove, D., & Lu, D. (2023). MLOps: Automatic, Zero-Touch and Reusable Machine Learning Training and Serving Pipelines. *2023 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS)*, 175–181. https://doi.org/10.1109/IoTaIS60147.2023.10346079

Tang, S.-T., Tjia, V., Noga, T., Febri, J., Lien, C.-Y., Chu, W.-C., Chen, C.-Y., & Hsiao, C.-H. (2023). Creating a Medical Imaging Workflow Based on FHIR, DICOMweb, and SVG. *Journal of Digital Imaging*, *36*(3), 794–803. https://doi.org/10.1007/s10278-021-00522-6

Theriault-Lauzier, P., Corbin, D., Tastet, O., Langlais, E. L., Taji, B., Kang, G., Chong, A.-Y., So, D., Tang, A., Gichoya, J. W., Chandar, S., Déziel, P.-L., Hussin, J. G., Kadoury, S., & Avram, R. (2024). A Responsible Framework for Applying Artificial Intelligence on Medical Images and Signals at the Point of Care: The PACS-AI Platform. *Canadian Journal of Cardiology*, *40*(10), 1828–1840. https://doi.org/10.1016/j.cjca.2024.05.025

Usman, M., Ferlin, S., Brunstrom, A., & Taheri, J. (2022). A Survey on Observability of Distributed Edge & Container-Based Microservices. *IEEE Access, 10,* 86904–86919. https://doi.org/10.1109/ACCESS.2022.3193102

ANEXOS

Pruebas en modo CPU → Modelo Tensorflow

Se realizó una prueba con un lote de 10 imágenes para ejecutarse con la GPU desactivada.

Tabla 17
Resultados de inferencia en CPU (Modelo TensorFlow)

Prueba	Tiempo (s)	Uso CPU (%)	RAM (GB)	Tamaño Imagen (MB
1	2.360	~5%	10.3	2,52 MB
2	1.580	~4%	10.3	2,37 MB
3	1.568	~4%	10.3	2,70 MB
4	1.568	~5%	10.6	2,52 MB
5	1.543	~7%	10.8	2,37 MB
6	1.533	~5%	10.8	2,70 MB
7	1.598	~6%	10.8	2,52 MB
8	1.594	~5%	10.7	2,37 MB
9	1.567	~4%	10.8	2,70 MB
10	1.665	~4%	10.9	2,52 MB

Nota. Estos son los resultados tras 10 ejecuciones consecutivas con GPU desactivada.

Promedio tiempo CPU: 1.58 segundos

Uso promedio CPU: 5.1 %

Consumo promedio RAM: 10.77 GB

Se realizaron diez ejecuciones consecutivas del microservicio en modo CPU. La primera ejecución registró un tiempo de 2.36 segundos debido a la carga inicial del modelo en memoria. A partir de la segunda ejecución, los tiempos se estabilizaron en un rango entre 1.53 y 1.66 segundos. El tiempo promedio final fue de 1.66 segundos por imagen.

El consumo de CPU se mantuvo entre 4% y 7%, mientras que el uso de memoria RAM permaneció estable alrededor de 10.7 -- 10.9 GB durante todas las ejecuciones. No se observaron incrementos progresivos en el consumo de recursos, lo que indica ausencia de fugas de memoria y estabilidad operativa del sistema.

Pruebas en modo GPU → Modelo **nnUNet**

Tabla 18
Resultados de inferencia en GPU (Modelo nnU-Net)

Nota. Se realizaron 10 ejecuciones consecutivas del modelo utilizando GPU

Prueba	Tiempo (s)	Uso GPU (%)	VRAM (GB)	Tamaño Imagen (MB
1	16.481	85	0.4	2,52 MB
2	17.787	92	0.4	2,37 MB
3	17.336	90	0.4	2,70 MB
4	11.563	90	0.4	2,52 MB
5	17.617	97	2.1	2,37 MB
6	17.606	43	0.6	2,70 MB
7	11.714	29	0.6	2,52 MB
8	17.521	62	0.6	2,37 MB
9	18.301	43	0.4	2,70 MB

10	10.624	21	0.6	2,52 MB

Promedio tiempo GPU: 15.455 segundos

Desviación estándar: 3.03 segundos

Uso promedio GPU: 60.9 %

Consumo promedio VRAM: 0.71 GB

La Tabla 15 presenta la comparación del rendimiento del modelo nnU-Net V2 ejecutado en modo CPU y GPU. Se observa que la aceleración mediante GPU reduce significativamente el tiempo promedio de inferencia, manteniendo un consumo de memoria estable y evidenciando un uso eficiente de VRAM durante el proceso de segmentación.

Como se observa en la Tabla 7, la ejecución en modo CPU presentó un tiempo promedio de 1.58 segundos, mientras que la ejecución en GPU mostró un tiempo promedio de 15.46 segundos.

Aunque la GPU alcanzó mayores niveles de utilización (65.2%) y uso de memoria dedicada (0.61 GB de VRAM), el tiempo total de inferencia fue superior al modo CPU en este entorno específico.

Esto sugiere que, para imágenes de tamaño moderado y bajo volumen de datos, la sobrecarga de inicialización y transferencia hacia GPU puede impactar negativamente el tiempo total de ejecución.

Tabla 19

Comparación de rendimiento entre los modelos UNet 3D MultiScale (TensorFlow) y nnU-Net V2 (PyTorch) bajo sus configuraciones óptimas de hardware

Métrica	UNet 3D MultiScale (TensorFlow / CPU)	nnU-Net V2 (PyTorch / GPU)
Parámetros entrenables	~5.64 millones	~88 millones
Hardware de ejecución	CPU (Intel i9-13980HX)	GPU (NVIDIA RTX 4060, 8 GB VRAM)
Tiempo promedio (s)	1.58 s	15.46 s
Desviación estándar (s)	~0.23 s	3.03 s
Uso promedio hardware (%)	5.1 % (CPU)	65.2 % (GPU)
RAM promedio (GB)	10.77 GB	12.8 GB
VRAM promedio (GB)	N/A	0.61 GB

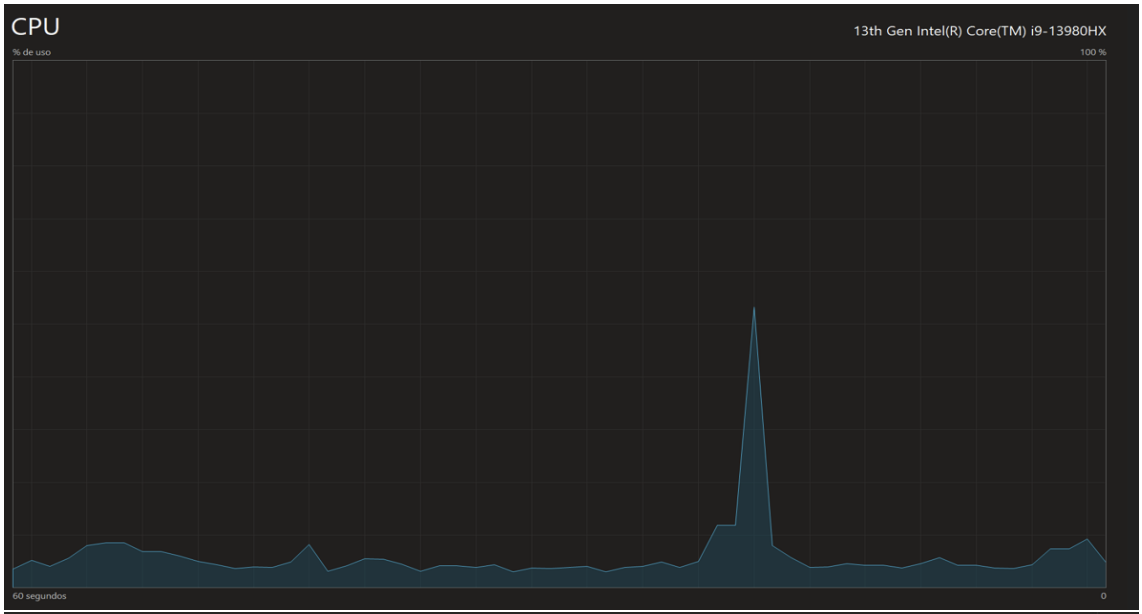
Nota. Cada modelo fue ejecutado en su configuración de hardware óptima. UNet 3D MultiScale opera eficientemente en CPU dado su menor complejidad arquitectónica. nnU-Net V2 requiere aceleración por GPU para operar dentro de rangos de latencia aceptables. Los valores corresponden al promedio de 10 ejecuciones consecutivas.

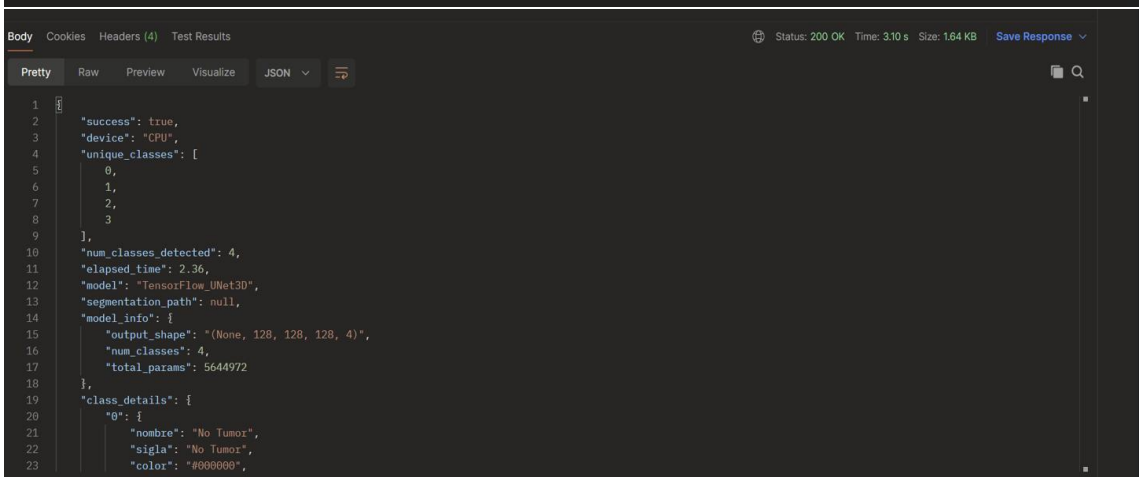
Durante las 10 ejecuciones consecutivas realizadas tanto en modo CPU (UNet 3D MultiScale) como en modo GPU (nnUNet), no se registraron errores de ejecución, fallos de segmentación ni interrupciones del sistema. Todas las solicitudes retornaron en estado HTTP 200 y confirmación de finalización exitosa.

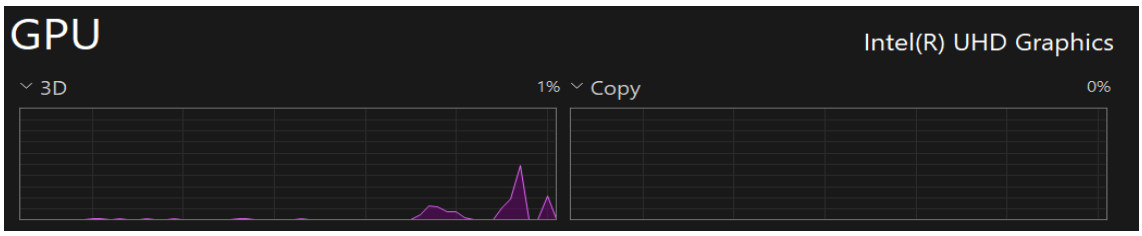
Esto representa una tasa de error del 0%, evidenciando estabilidad y robustez del sistema bajo condiciones controladas.

Pruebas en modo CPU → Modelo Tensorflow

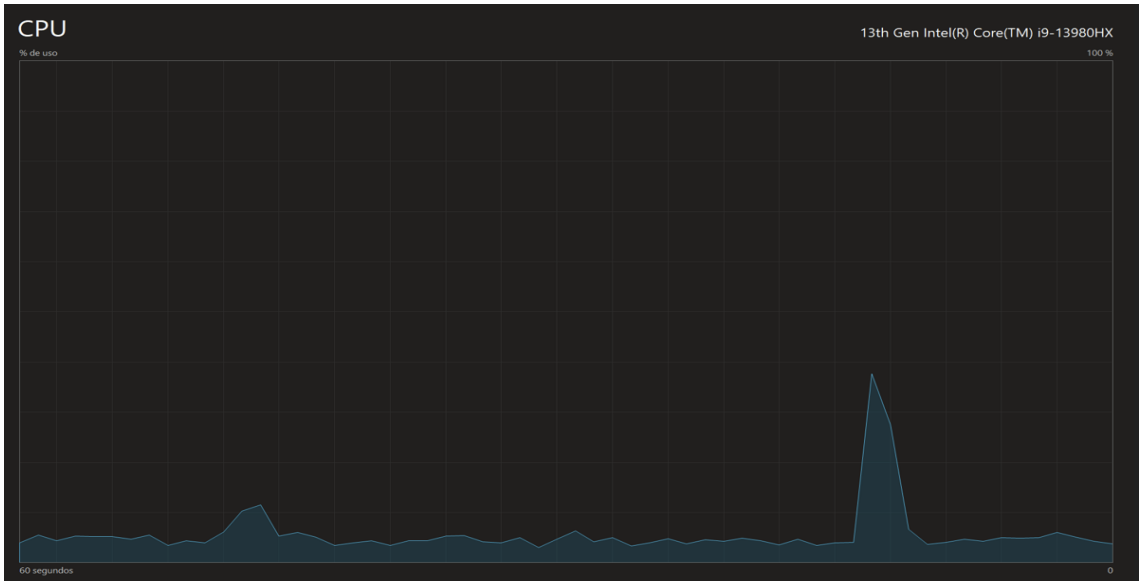
Prueba 1:

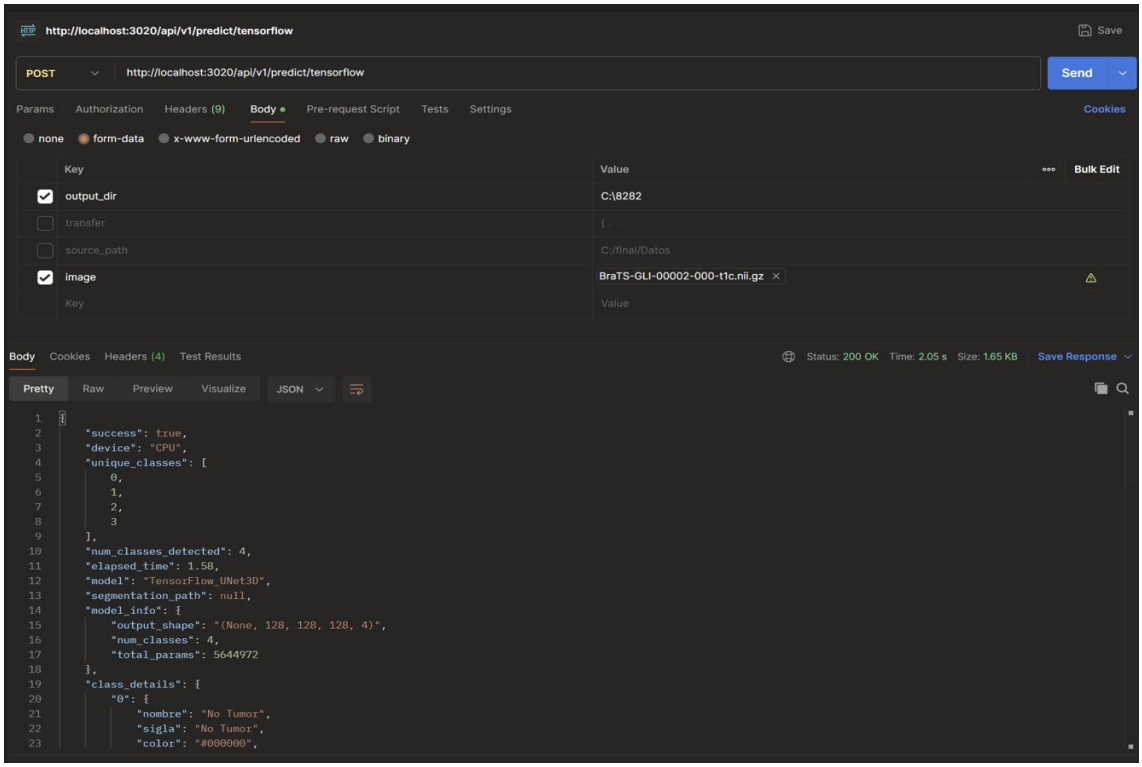


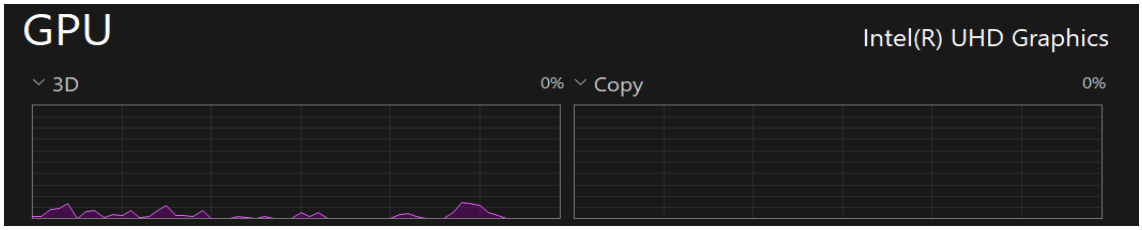




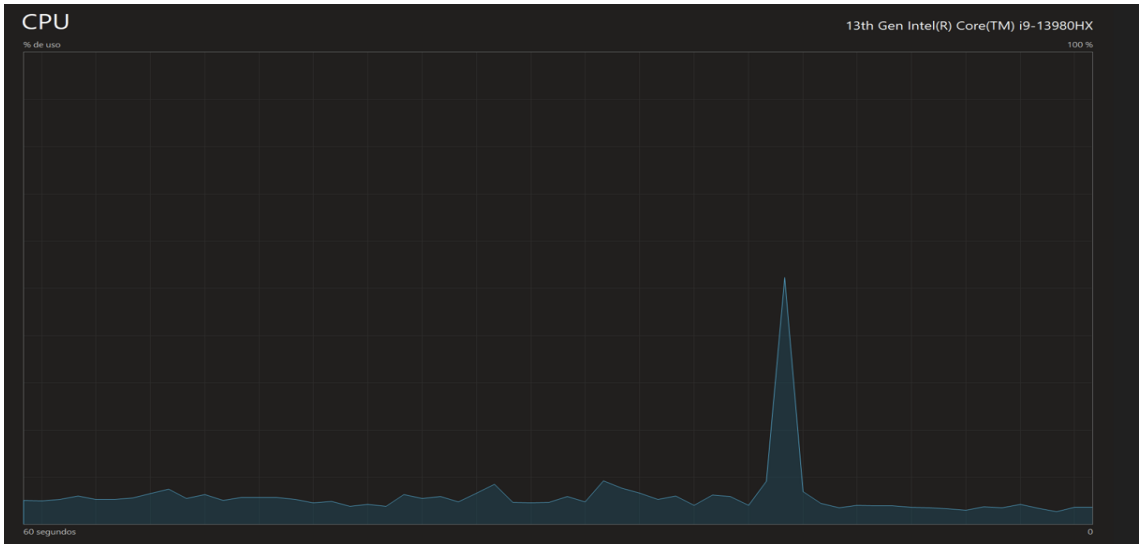
Prueba 2:

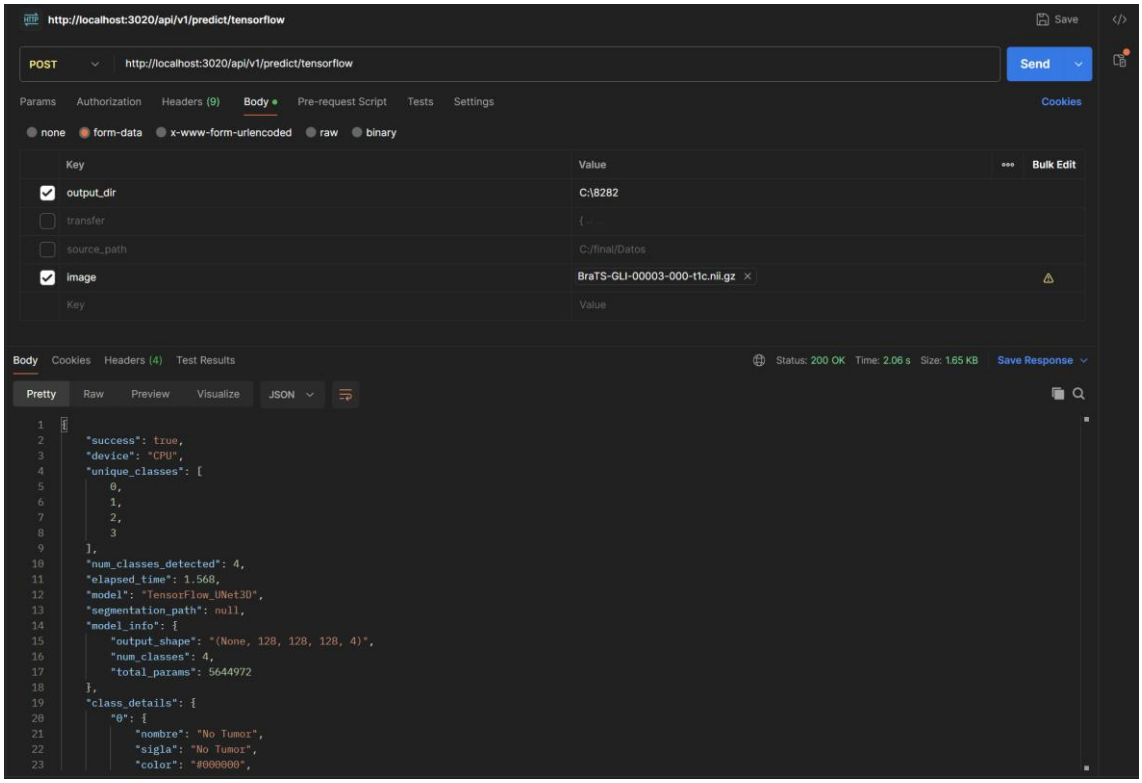

CPU
13th Gen Intel(R) Core(TM) i9-13980HX
% de uso
100 %
60 segundos
0

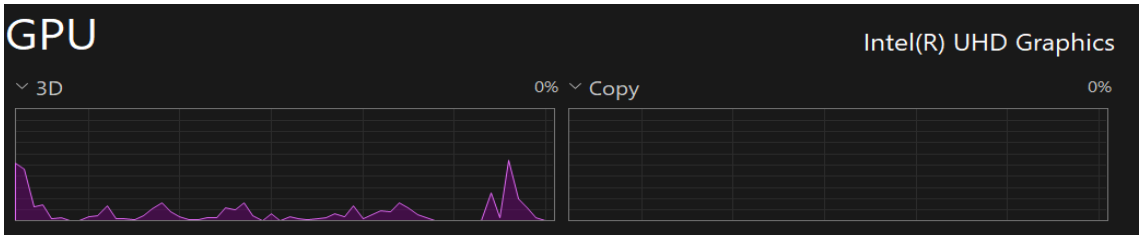

http://localhost:3020/api/v1/predict/tensorflow
Save
POST
http://localhost:3020/api/v1/predict/tensorflow
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00002-000-t1c.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 2.05 s
Size: 1.65 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"device": "CPU",
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"elapsed_time": 1.58,
"model": "TensorFlow_UNet3D",
"segmentation_path": null,
"model_info": {
"output_shape": "(None, 128, 128, 128, 4)",
"num_classes": 4,
"total_params": 5644972
},
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",


GPU
Intel(R) UHD Graphics
3D
0%
Copy
0%

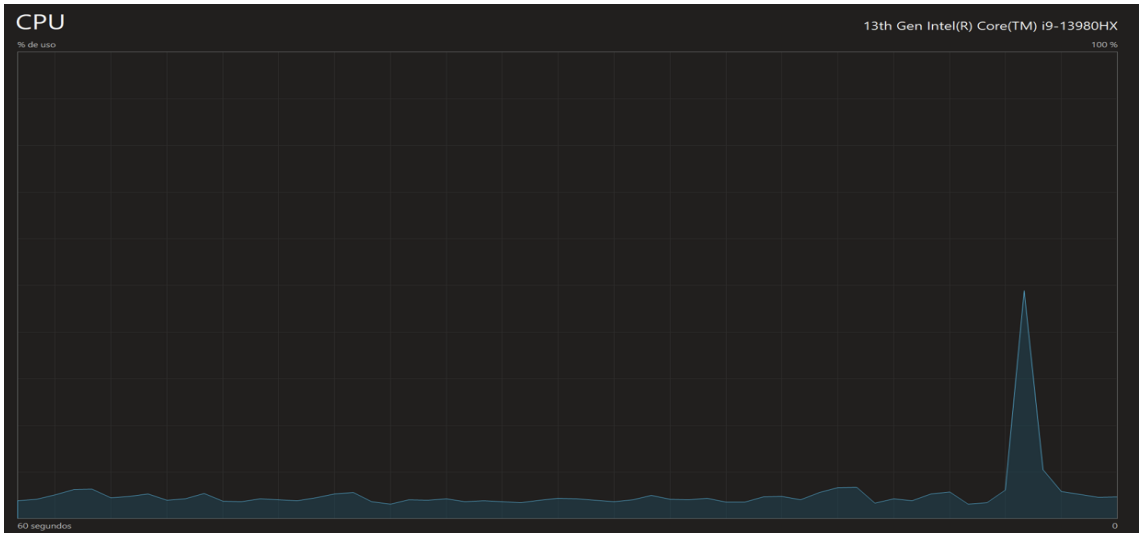
Prueba 3:

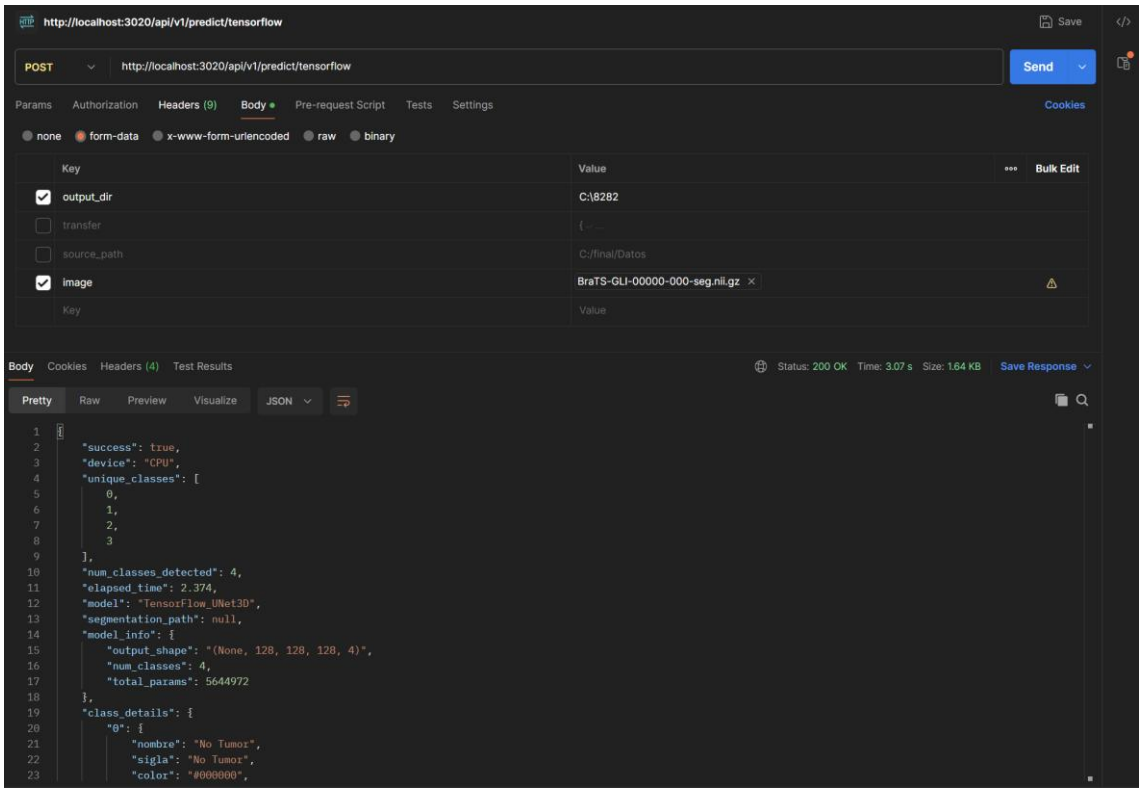

CPU
13th Gen Intel(R) Core(TM) i9-13980HX
% de uso
100 %
60 segundos
0

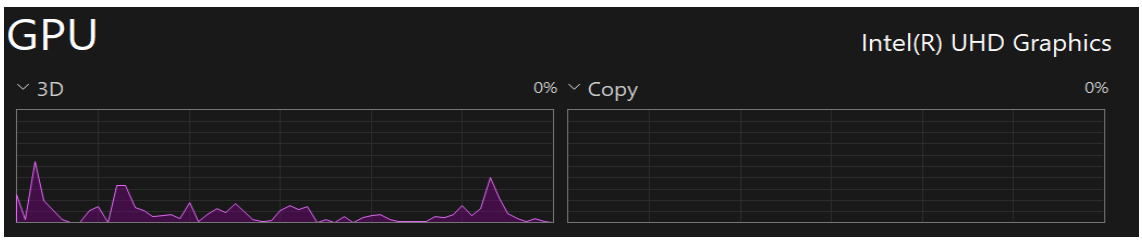

http://localhost:3020/api/v1/predict/tensorflow
Save
POST
http://localhost:3020/api/v1/predict/tensorflow
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00003-000-t1c.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 2.06 s
Size: 1.65 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"device": "CPU",
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"elapsed_time": 1.568,
"model": "TensorFlow_UNet3D",
"segmentation_path": null,
"model_info": {
"output_shape": "(None, 128, 128, 128, 4)",
"num_classes": 4,
"total_params": 5644972
},
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",


GPU
Intel(R) UHD Graphics
3D
0%
Copy
0%

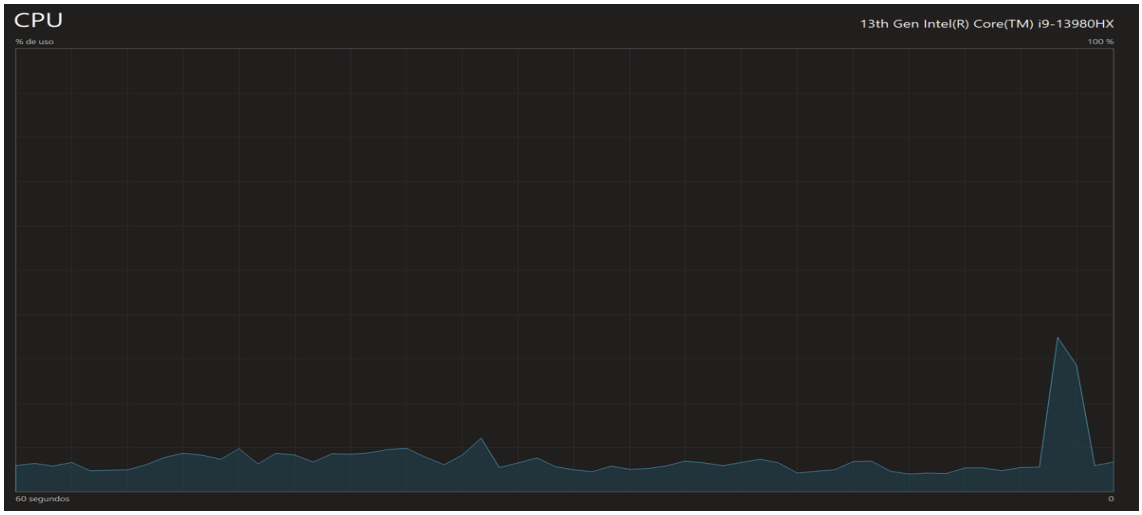
Prueba 4:

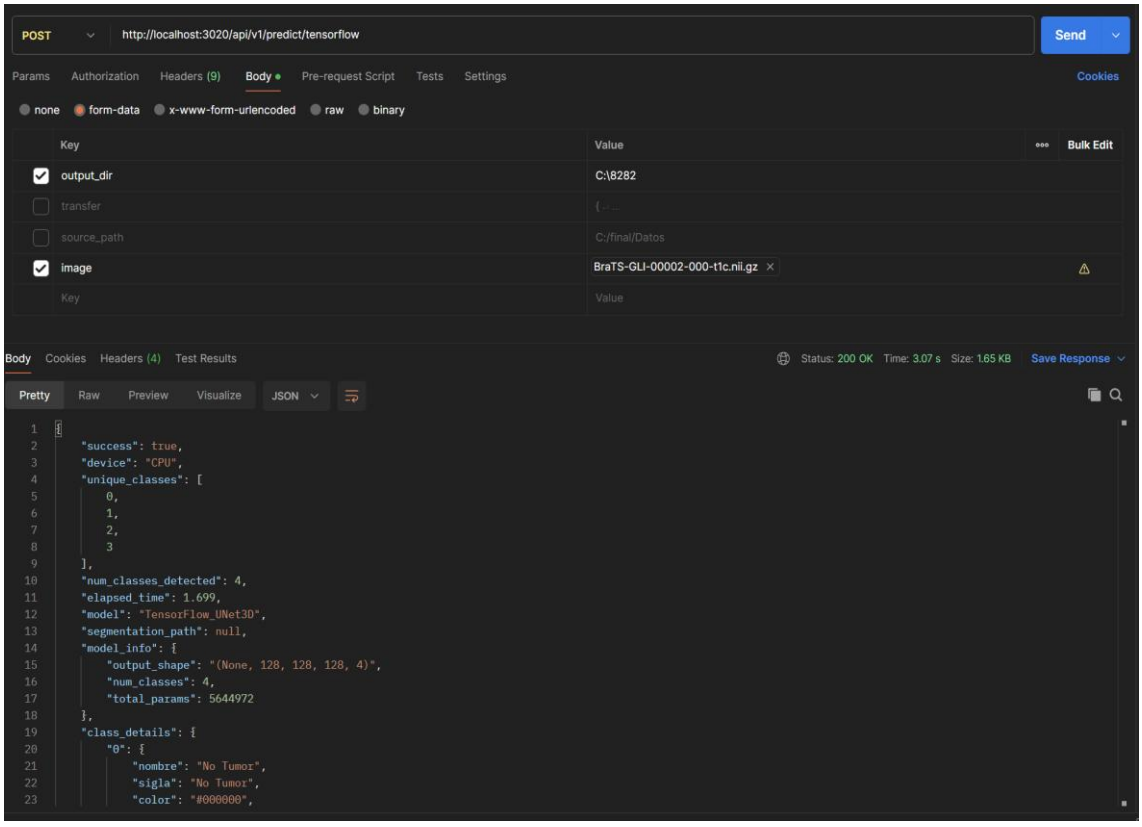

CPU
13th Gen Intel(R) Core(TM) i9-13980HX
% de uso
100 %
60 segundos
0

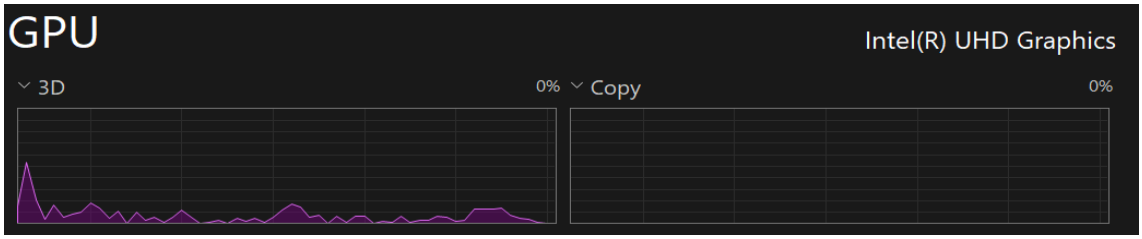

http://localhost:3020/api/v1/predict/tensorflow
Save
POST
http://localhost:3020/api/v1/predict/tensorflow
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00000-000-seg.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 3.07 s
Size: 1.64 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"device": "CPU",
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"elapsed_time": 2.374,
"model": "TensorFlow_UNet3D",
"segmentation_path": null,
"model_info": {
"output_shape": "(None, 128, 128, 128, 4)",
"num_classes": 4,
"total_params": 5644972
},
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",


GPU
Intel(R) UHD Graphics
3D
0%
Copy
0%

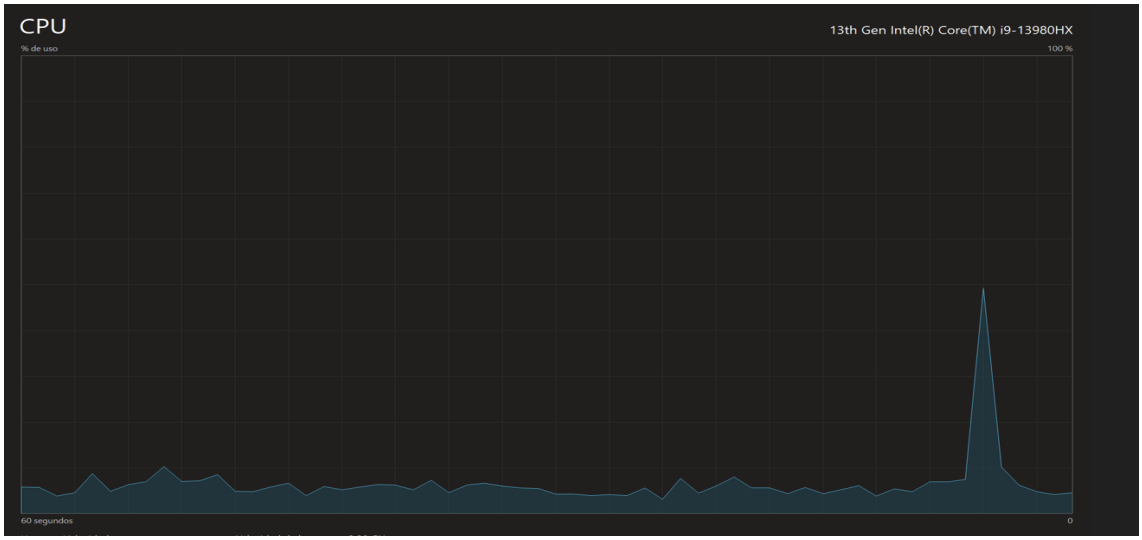
Prueba 5:

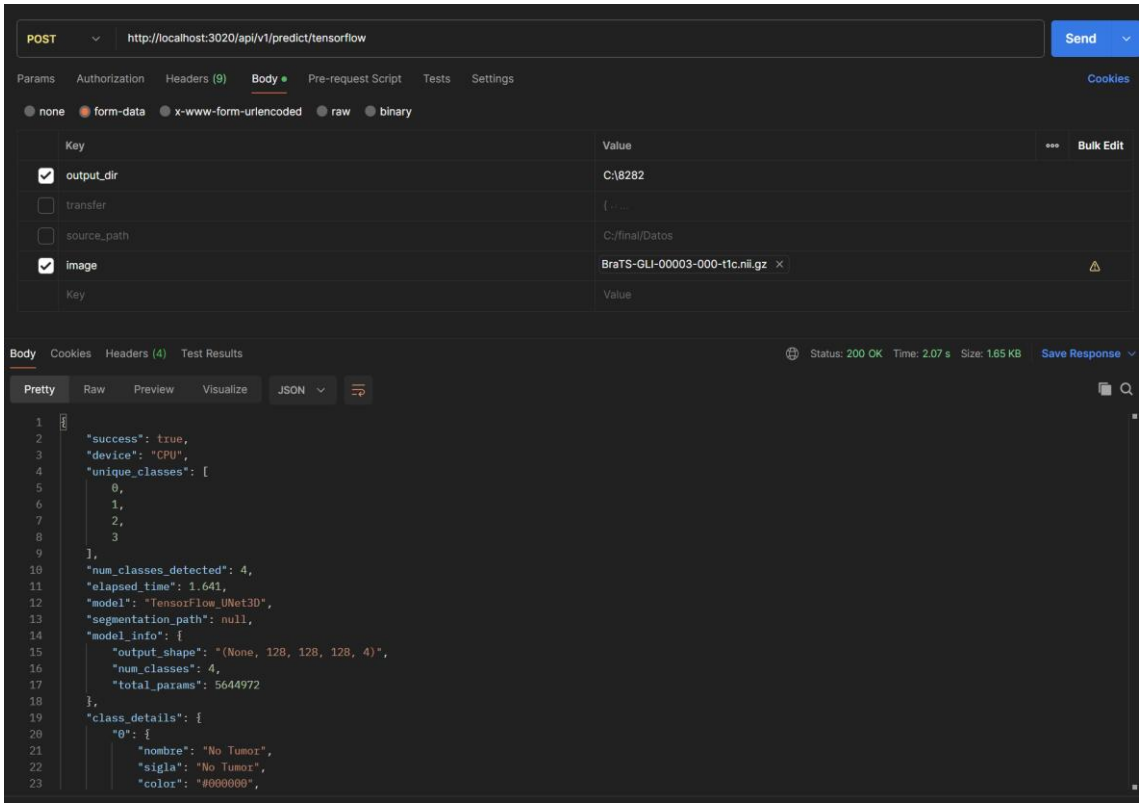

CPU
13th Gen Intel(R) Core(TM) i9-13980HX
% de uso
100 %
60 segundos
0

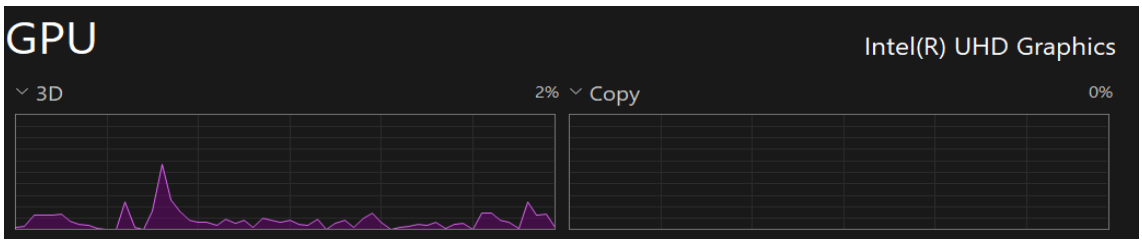

POST
http://localhost:3020/api/v1/predict/tensorflow
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00002-000-t1c.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 3.07 s
Size: 1.65 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"device": "CPU",
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"elapsed_time": 1.699,
"model": "TensorFlow_UNet3D",
"segmentation_path": null,
"model_info": {
"output_shape": "(None, 128, 128, 128, 4)",
"num_classes": 4,
"total_params": 5644972
},
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",


GPU
Intel(R) UHD Graphics
3D
0%
Copy
0%

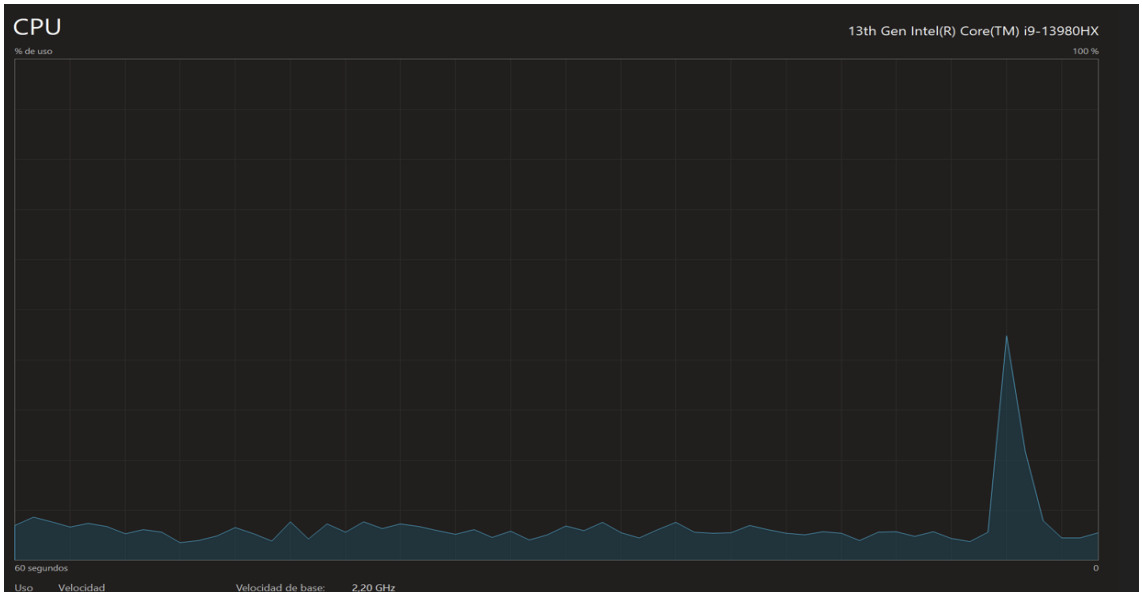
Prueba 6:

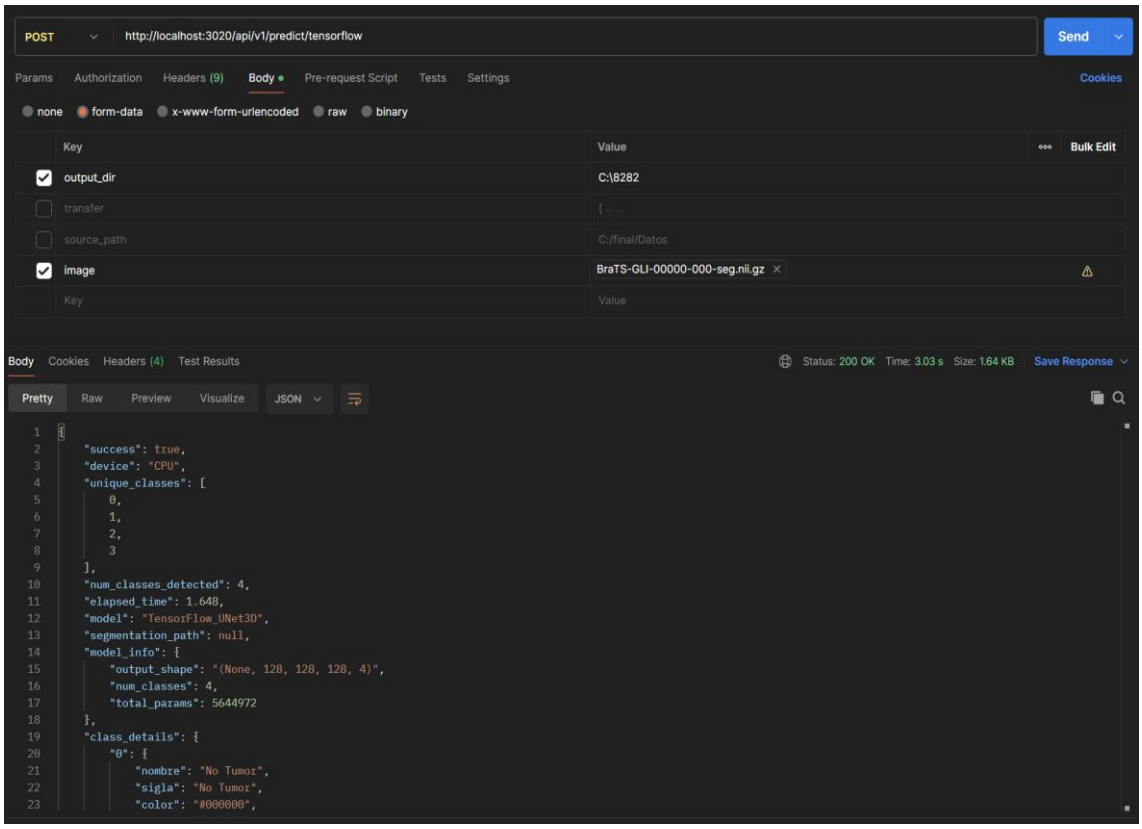

CPU
13th Gen Intel(R) Core(TM) i9-13980HX
% de uso
100 %
60 segundos
0

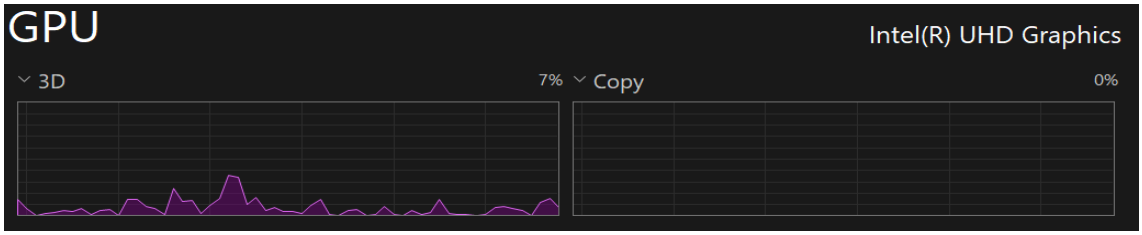

POST
http://localhost:3020/api/v1/predict/tensorflow
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00003-000-t1c.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 2.07 s
Size: 1.65 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"device": "CPU",
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"elapsed_time": 1.641,
"model": "TensorFlow_UNet3D",
"segmentation_path": null,
"model_info": {
"output_shape": "(None, 128, 128, 128, 4)",
"num_classes": 4,
"total_params": 5644972
},
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",


GPU
Intel(R) UHD Graphics
3D
2%
Copy
0%

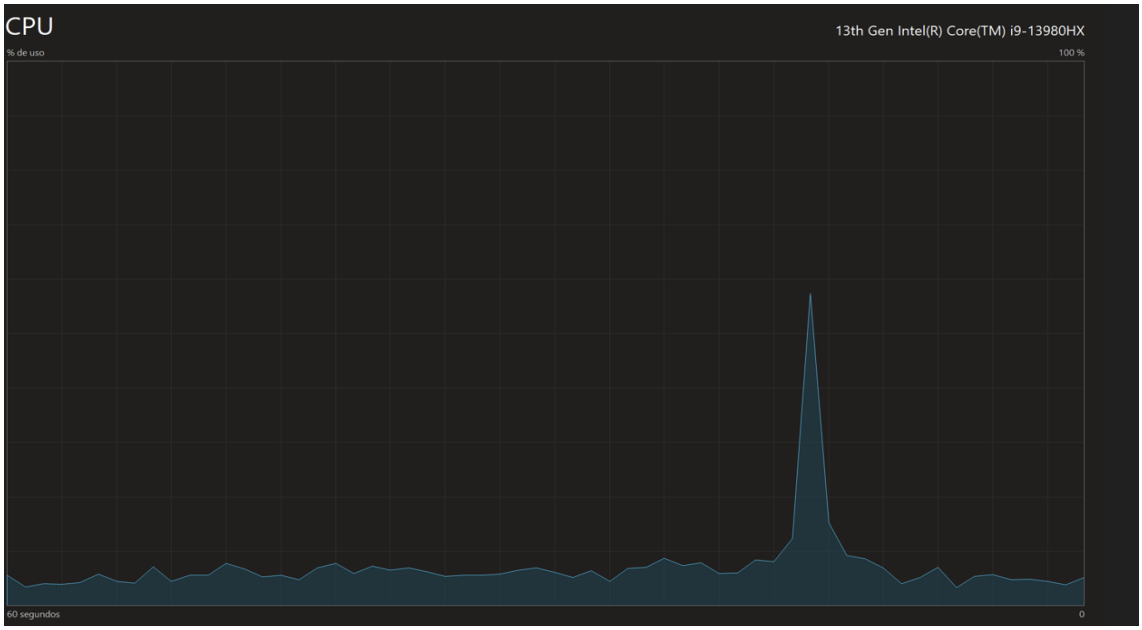
Prueba 7:

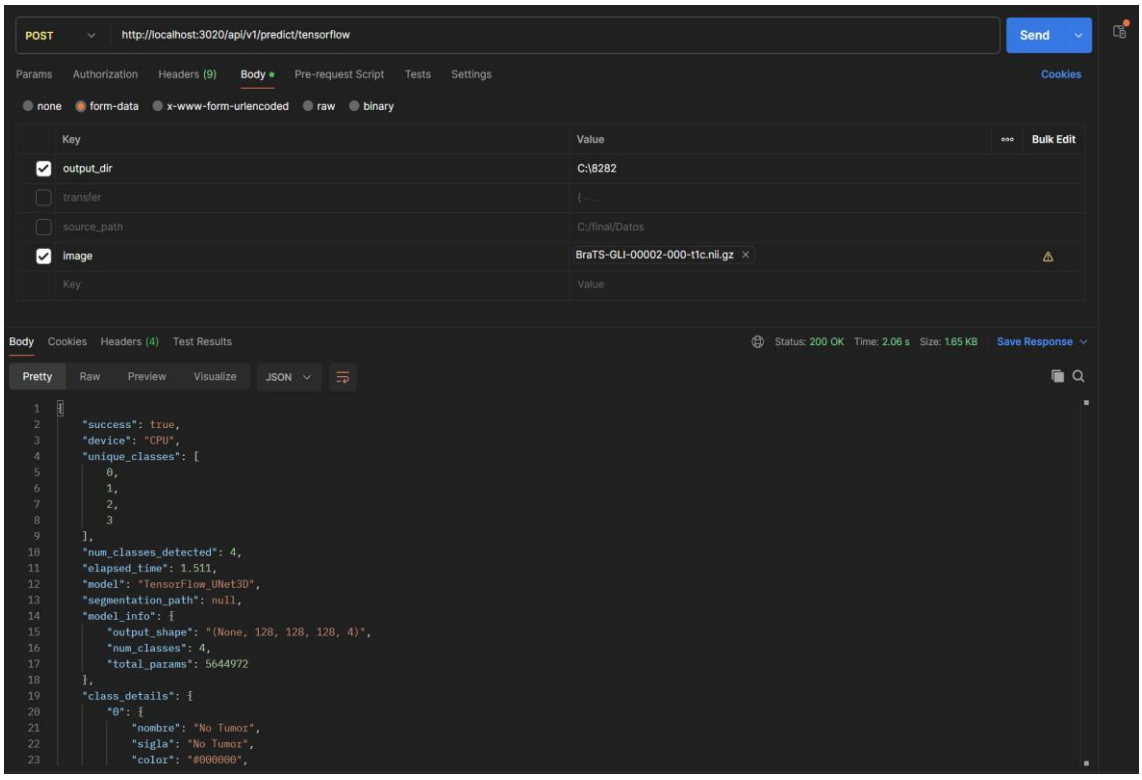

CPU
13th Gen Intel(R) Core(TM) i9-13980HX
% de uso
100 %
60 segundos
0
Uso
Velocidad
Velocidad de base: 2,20 GHz

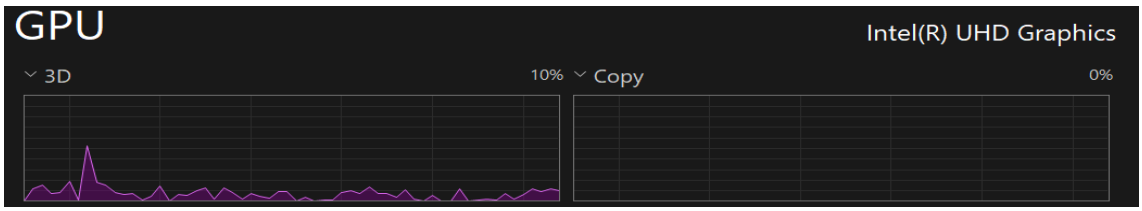

POST
http://localhost:3020/api/v1/predict/tensorflow
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00000-000-seg.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 3.03 s
Size: 1.64 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"device": "CPU",
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"elapsed_time": 1.648,
"model": "TensorFlow_UNet3D",
"segmentation_path": null,
"model_info": {
"output_shape": "(None, 128, 128, 128, 4)",
"num_classes": 4,
"total_params": 5644972
},
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",


GPU
Intel(R) UHD Graphics
3D
7%
Copy
0%

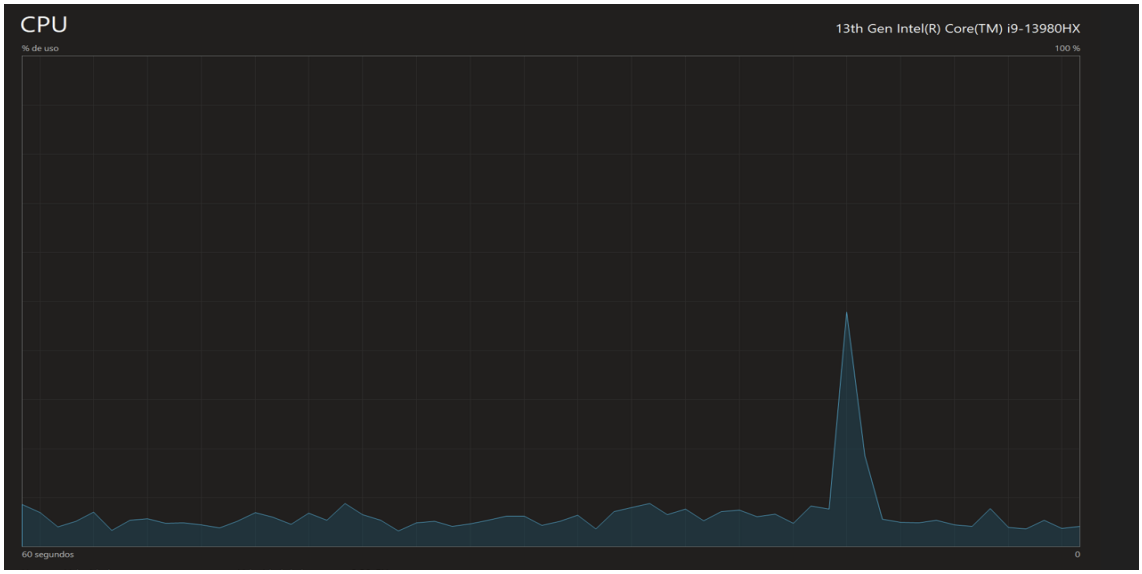
Prueba 8:

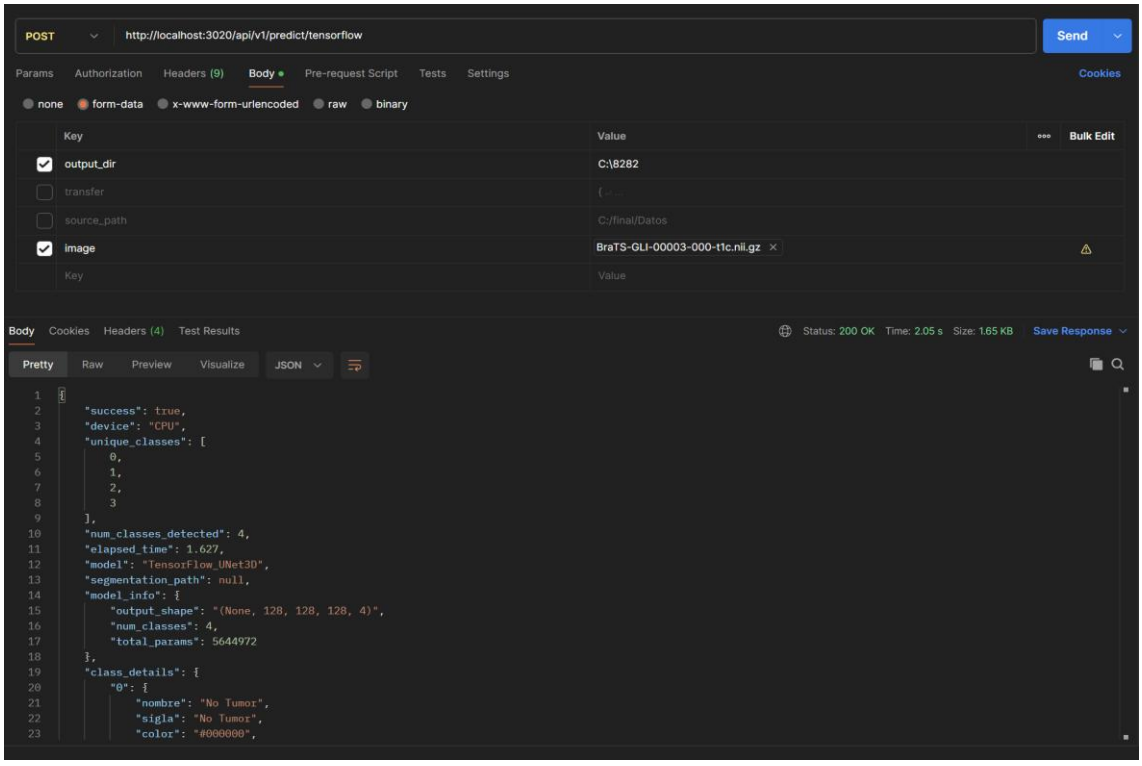

CPU
13th Gen Intel(R) Core(TM) i9-13980HX
% de uso
100 %
60 segundos
0

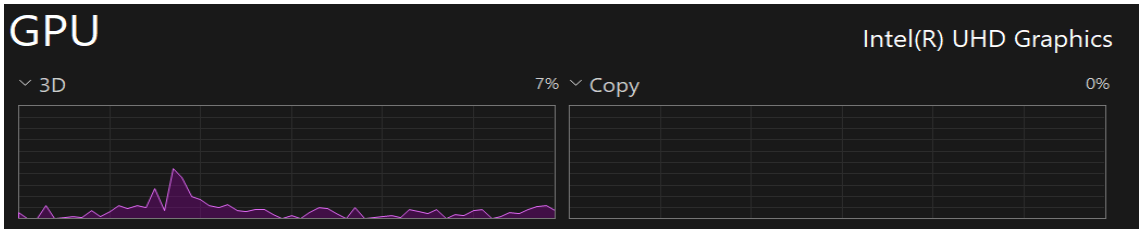

POST
http://localhost:3020/api/v1/predict/tensorflow
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00002-000-t1c.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 2.06 s
Size: 1.65 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"device": "CPU",
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"elapsed_time": 1.511,
"model": "TensorFlow_UNet3D",
"segmentation_path": null,
"model_info": {
"output_shape": "(None, 128, 128, 128, 4)",
"num_classes": 4,
"total_params": 5644972
},
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",


GPU
Intel(R) UHD Graphics
3D
10%
Copy
0%

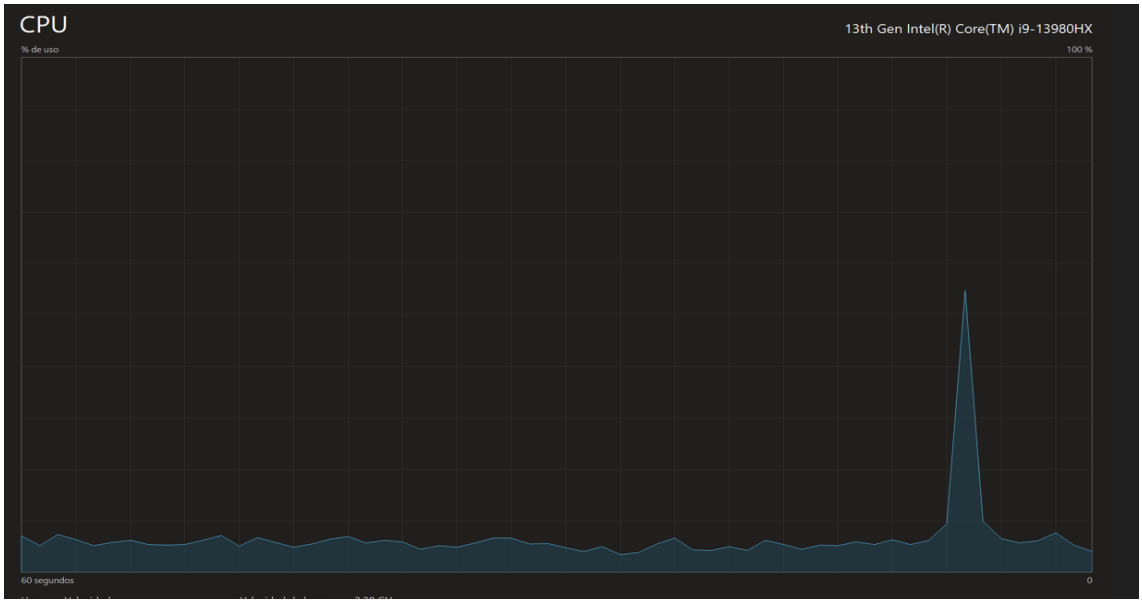
Prueba 9:

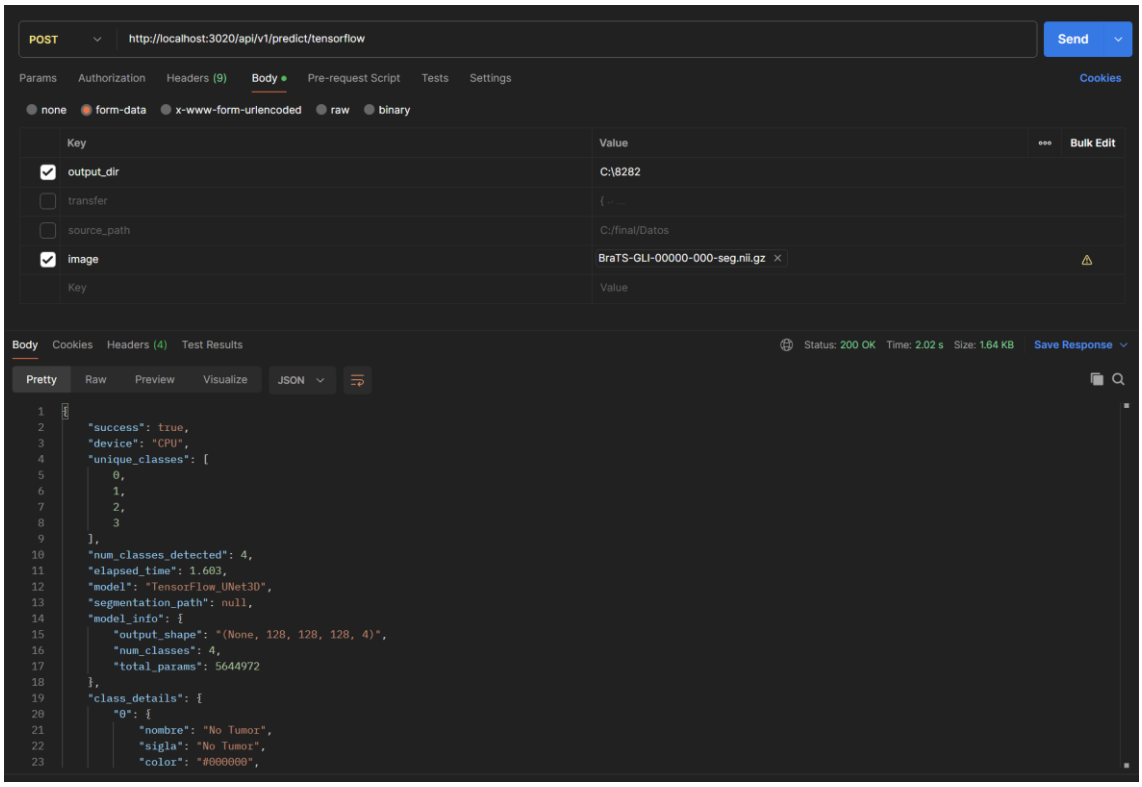

CPU
13th Gen Intel(R) Core(TM) i9-13980HX
% de uso
100 %
60 segundos
0

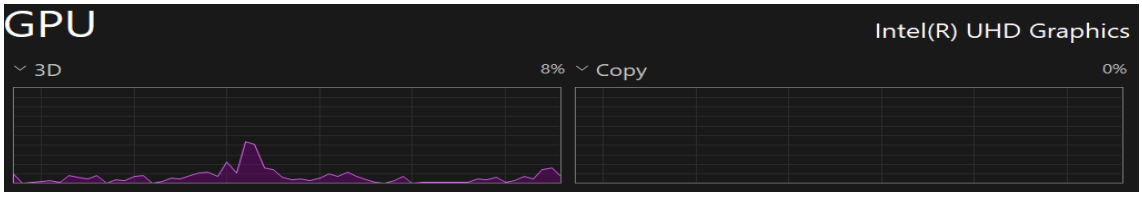

POST
http://localhost:3020/api/v1/predict/tensorflow
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00003-000-t1c.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 2.05 s
Size: 1.65 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"device": "CPU",
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"elapsed_time": 1.627,
"model": "TensorFlow_UNet3D",
"segmentation_path": null,
"model_info": {
"output_shape": "(None, 128, 128, 128, 4)",
"num_classes": 4,
"total_params": 5644972
},
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",


GPU
Intel(R) UHD Graphics
3D
7%
Copy
0%

Prueba 10:

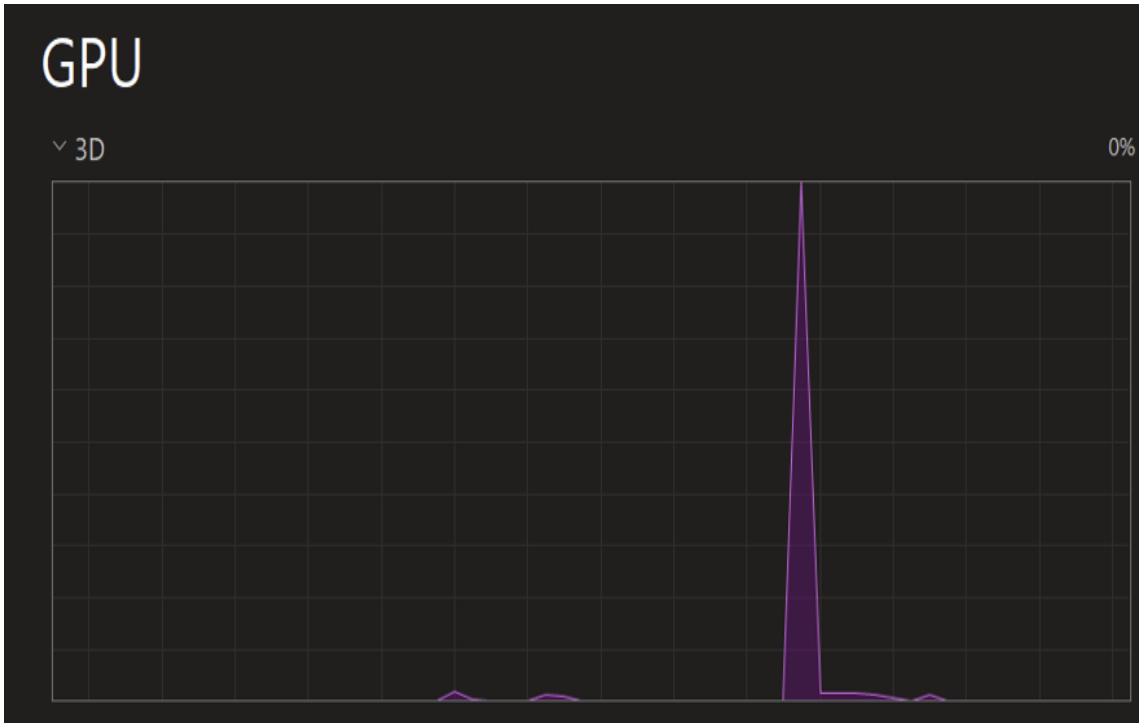

CPU
13th Gen Intel(R) Core(TM) i9-13980HX
% de uso
100 %
60 segundos
0

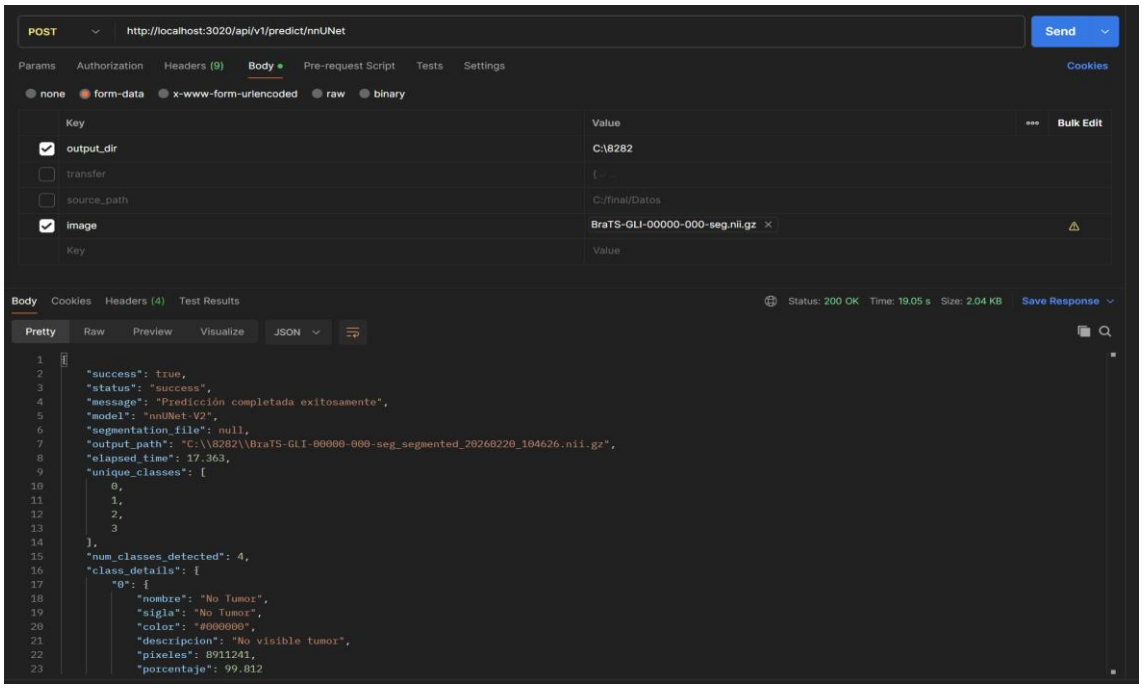

POST
http://localhost:3020/api/v1/predict/tensorflow
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00000-000-seg.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 2.02 s
Size: 1.64 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"device": "CPU",
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"elapsed_time": 1.603,
"model": "TensorFlow_UNet3D",
"segmentation_path": null,
"model_info": {
"output_shape": "(None, 128, 128, 128, 4)",
"num_classes": 4,
"total_params": 5644972
},
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",


GPU
Intel(R) UHD Graphics
3D
8%
Copy
0%

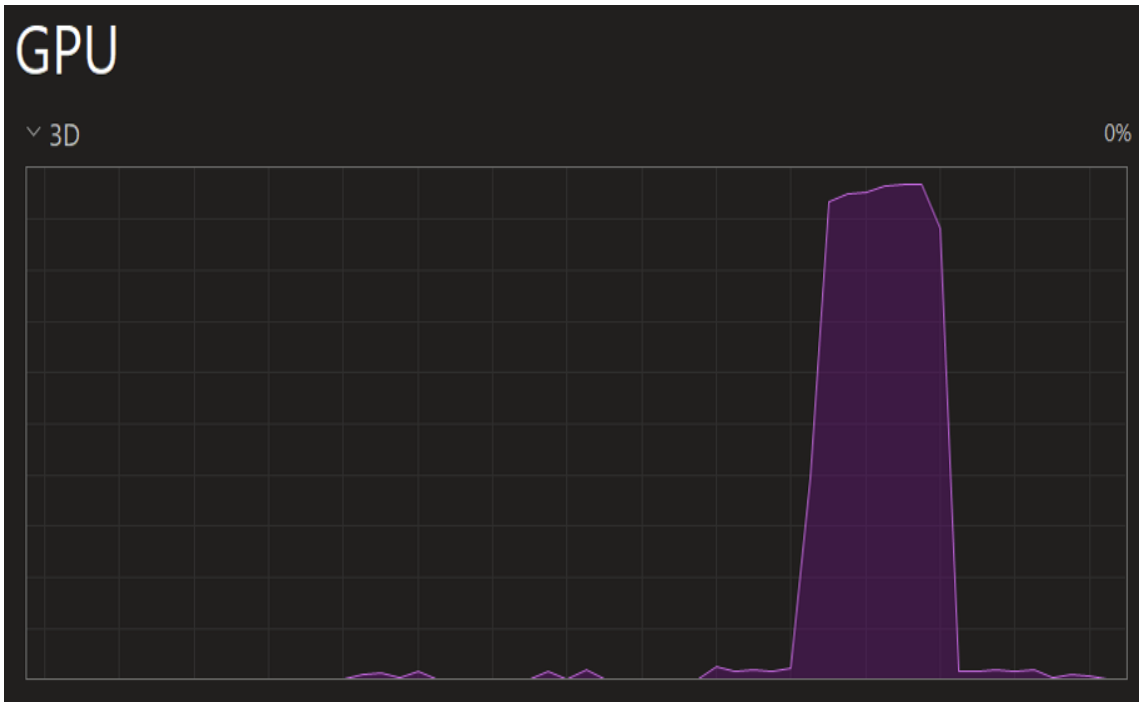
Pruebas en modo GPU → Modelo **nnUNet**

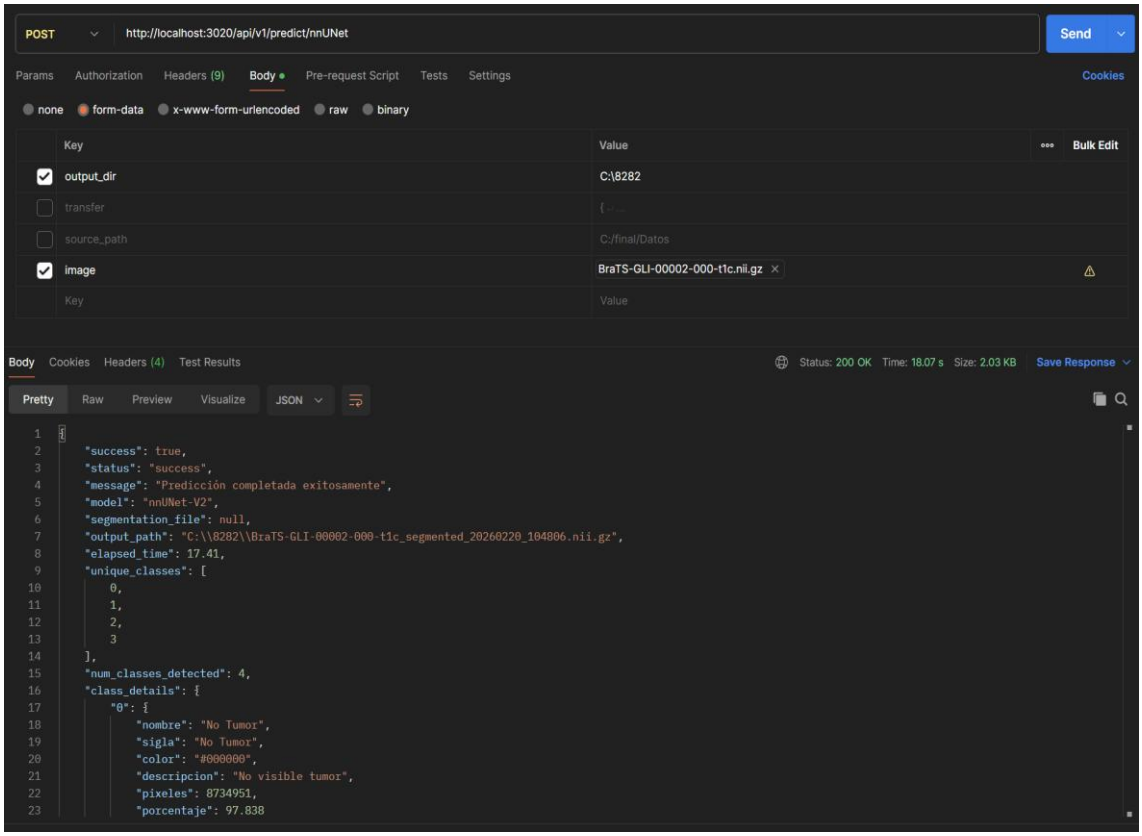
Prueba 1:



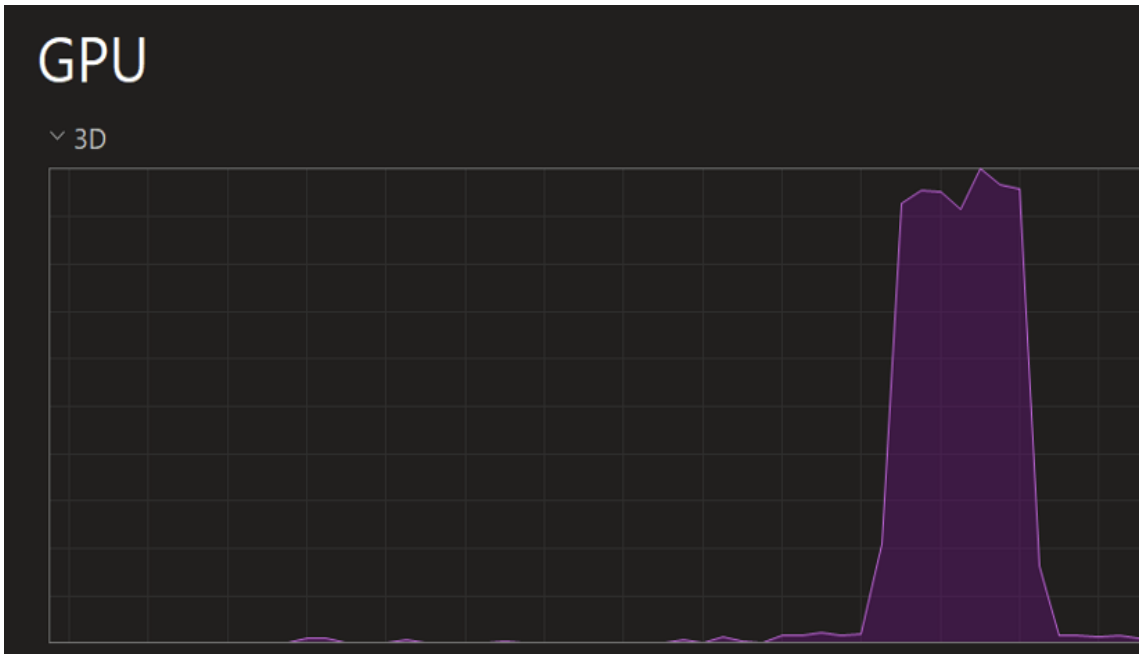


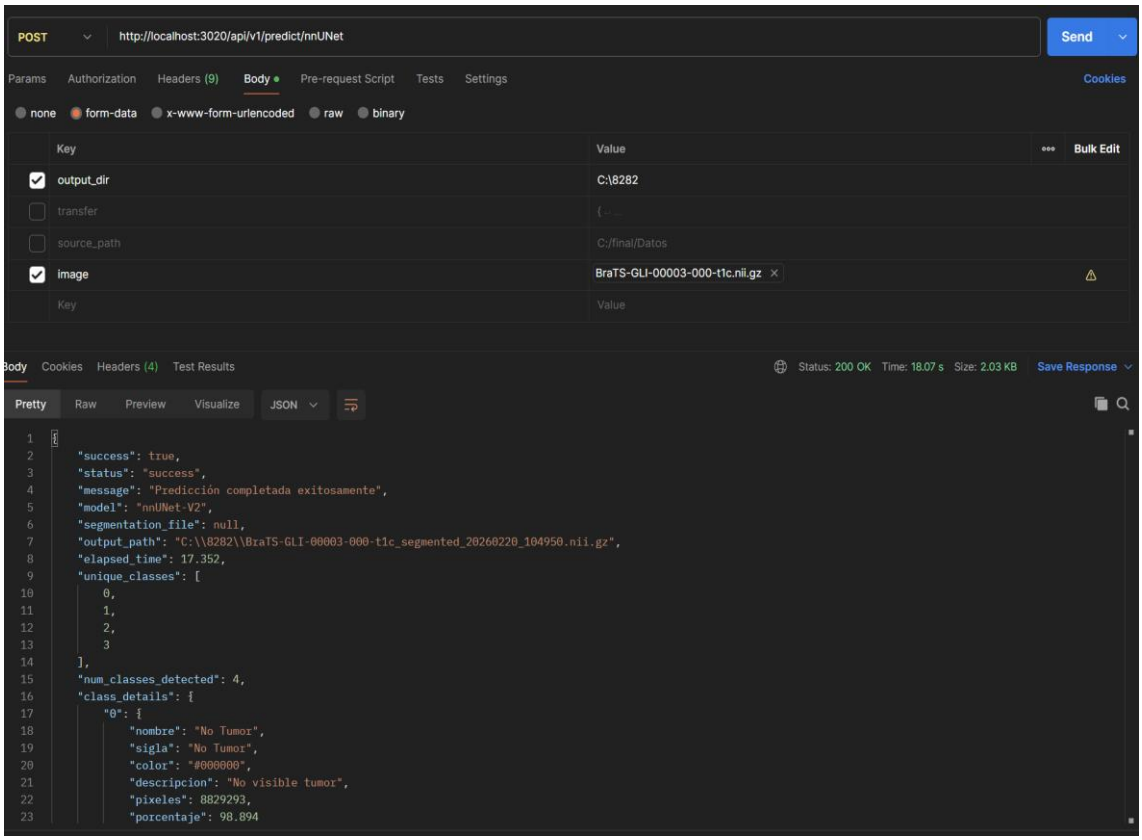
Prueba 2:


GPU
3D
0%

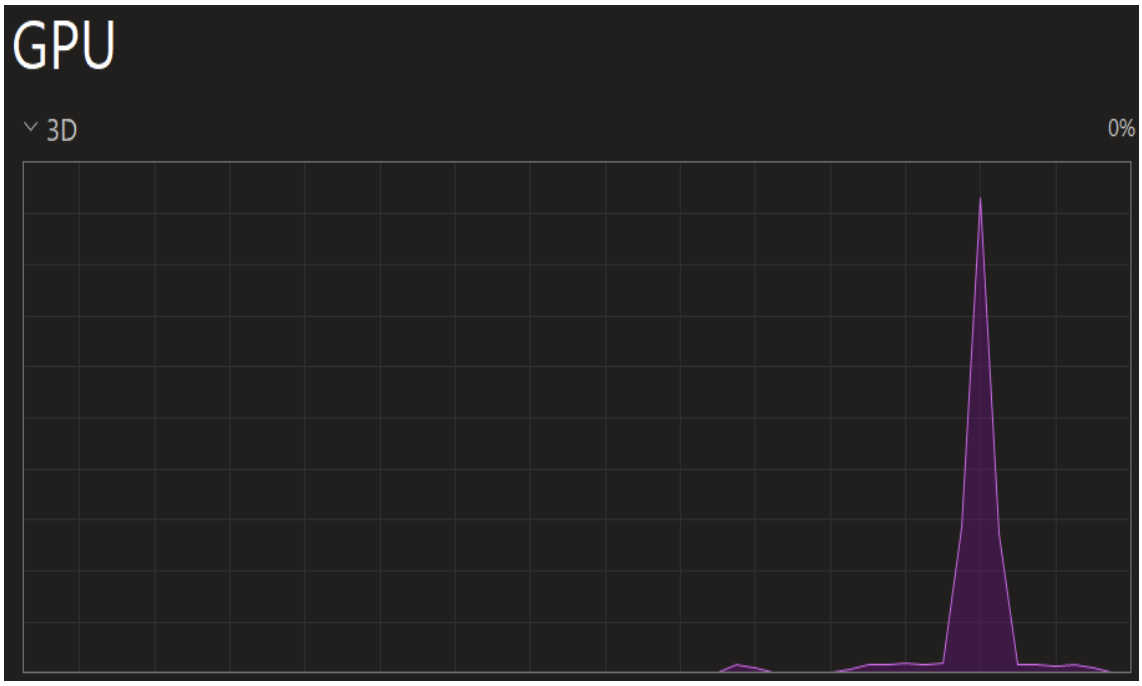

POST
http://localhost:3020/api/v1/predict/nnUNet
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00002-000-t1c.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 18.07 s
Size: 2.03 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"status": "success",
"message": "Predicción completada exitosamente",
"model": "nnUNet-V2",
"segmentation_file": null,
"output_path": "C:\\8282\\BraTS-GLI-00002-000-t1c_segmented_20260220_104806.nii.gz",
"elapsed_time": 17.41,
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",
"descripcion": "No visible tumor",
"pixeles": 8734951,
"porcentaje": 97.838

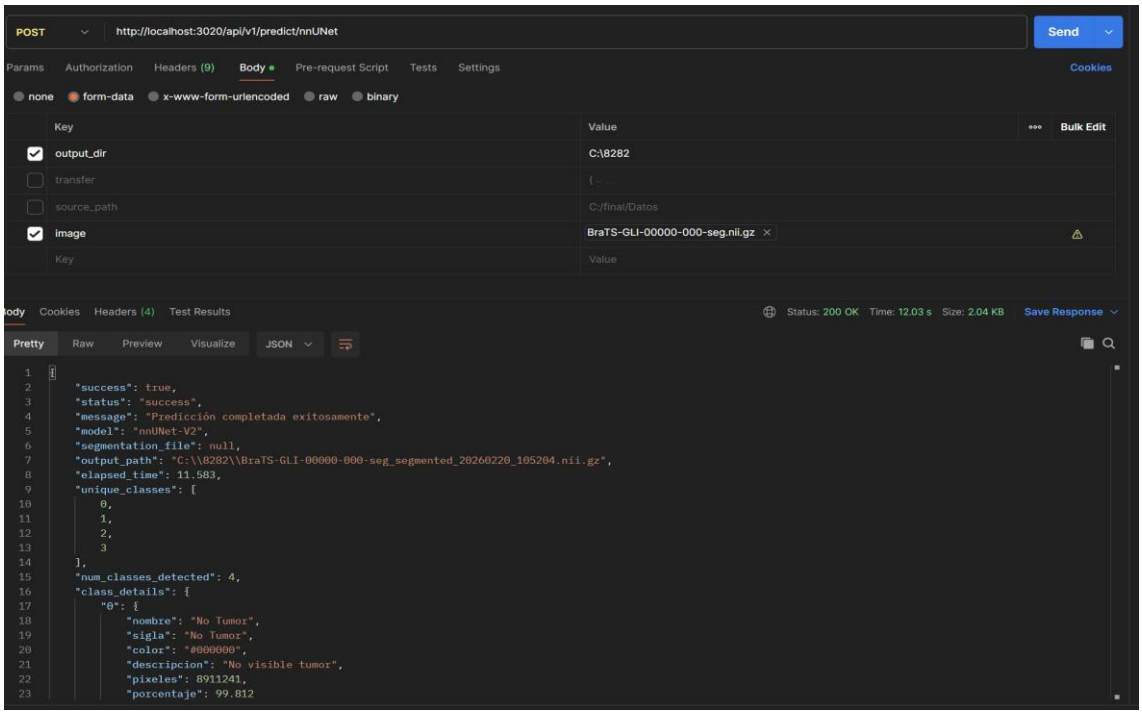
Prueba 3:


GPU
3D

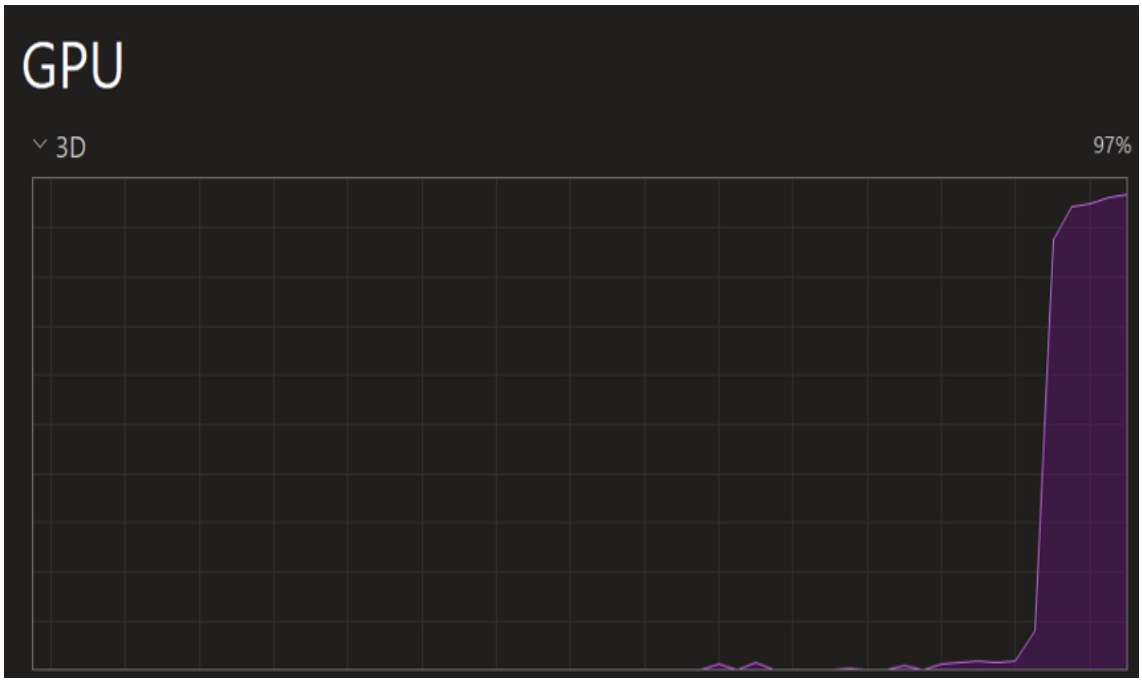

POST
http://localhost:3020/api/v1/predict/nnUNet
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00003-000-t1c.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 18.07 s
Size: 2.03 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"status": "success",
"message": "Predicción completada exitosamente",
"model": "nnUNet-V2",
"segmentation_file": null,
"output_path": "C:\\8282\\BraTS-GLI-00003-000-t1c_segmented_20260220_104950.nii.gz",
"elapsed_time": 17.352,
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",
"descripcion": "No visible tumor",
"pixeles": 8829293,
"porcentaje": 98.894

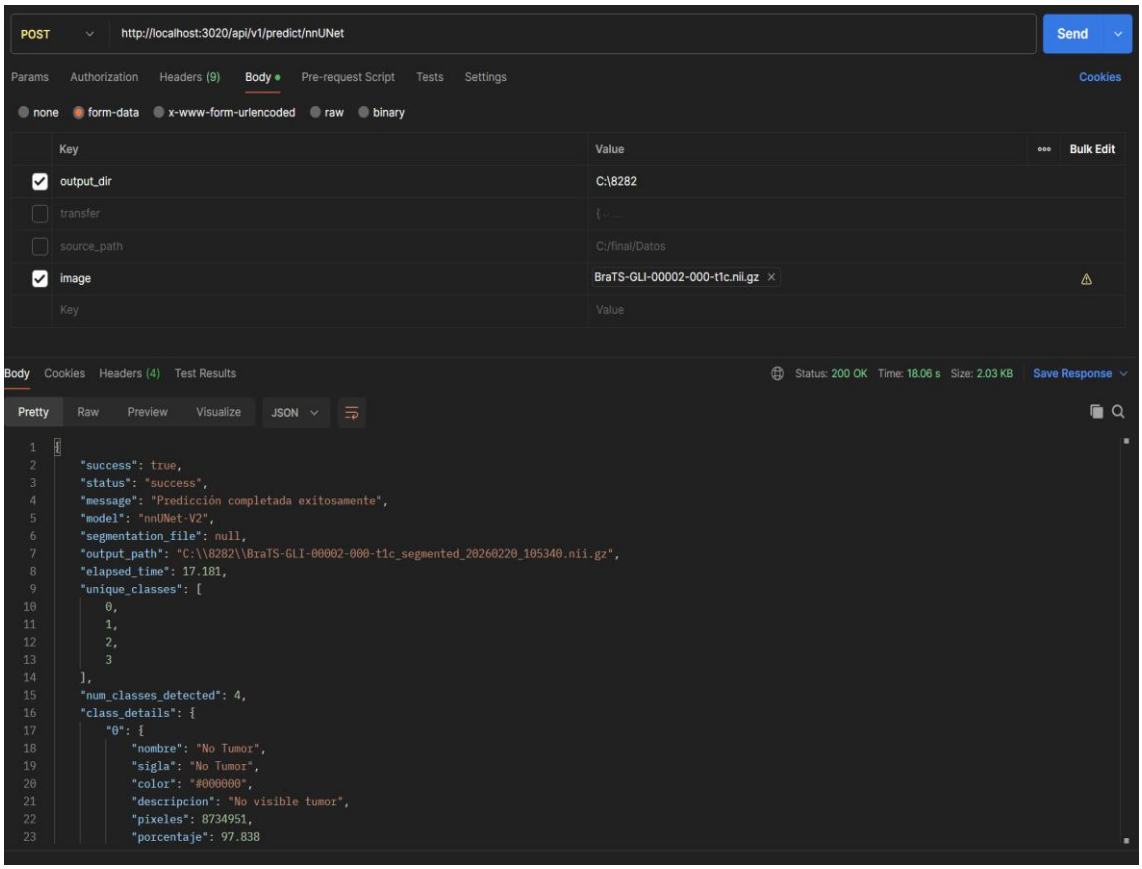
Prueba 4:


GPU
3D
0%

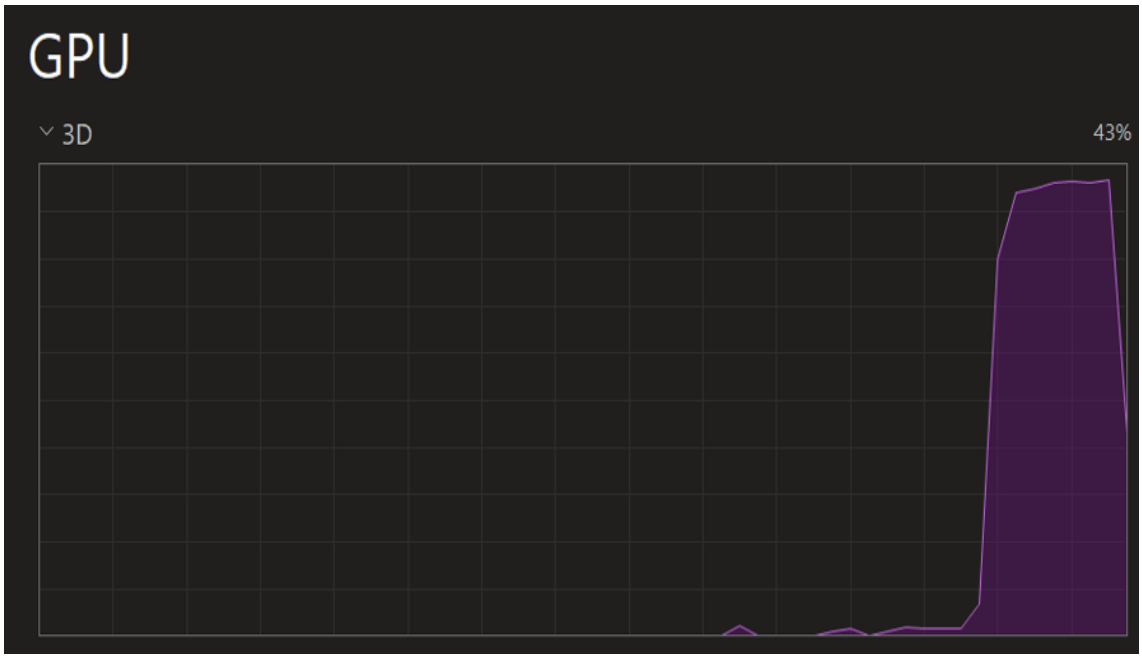

POST
http://localhost:3020/api/v1/predict/nnUNet
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00000-000-seg.nii.gz
Key
Value
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 12.03 s
Size: 2.04 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"status": "success",
"message": "Predicción completada exitosamente",
"model": "nnUNet-V2",
"segmentation_file": null,
"output_path": "C:\\8282\\BraTS-GLI-00000-000-seg_segmented_20260220_105204.nii.gz",
"elapsed_time": 11.583,
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",
"descripcion": "No visible tumor",
"pixeles": 8911241,
"porcentaje": 99.812

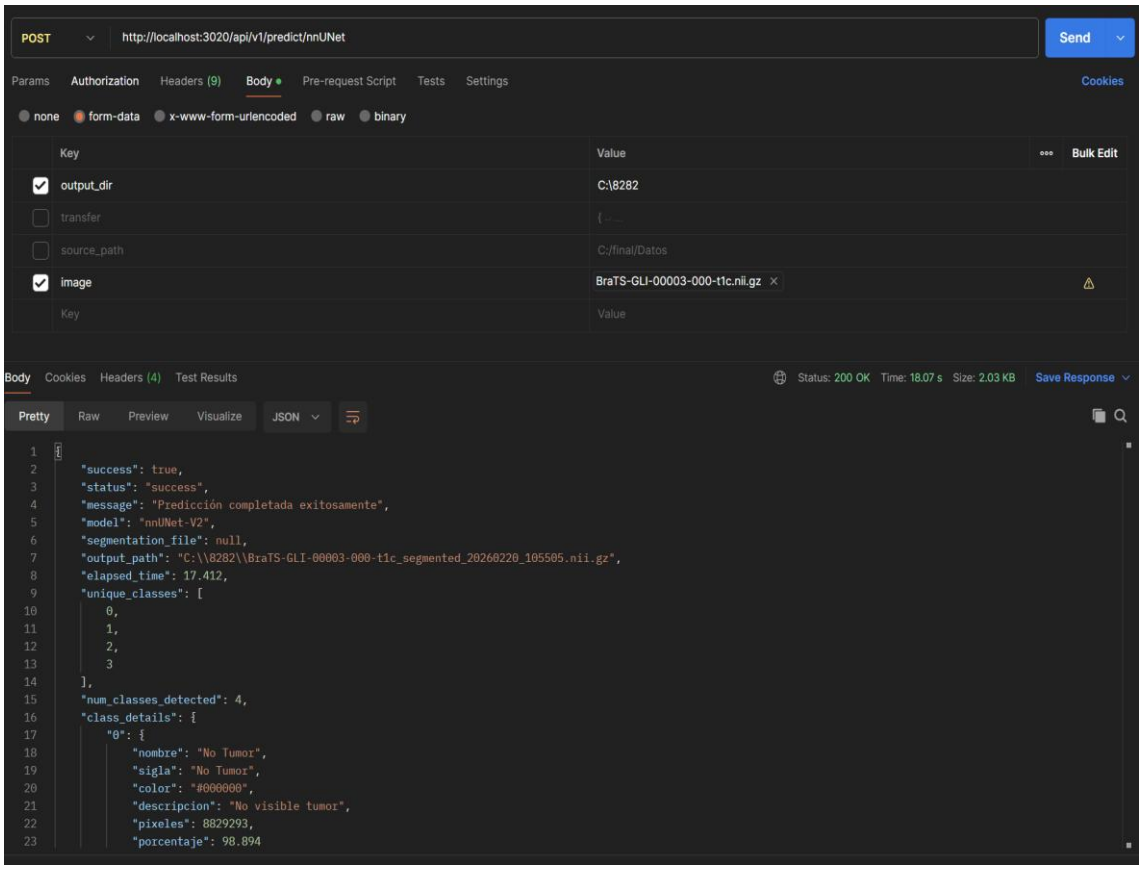
Prueba 5:


GPU
3D
97%

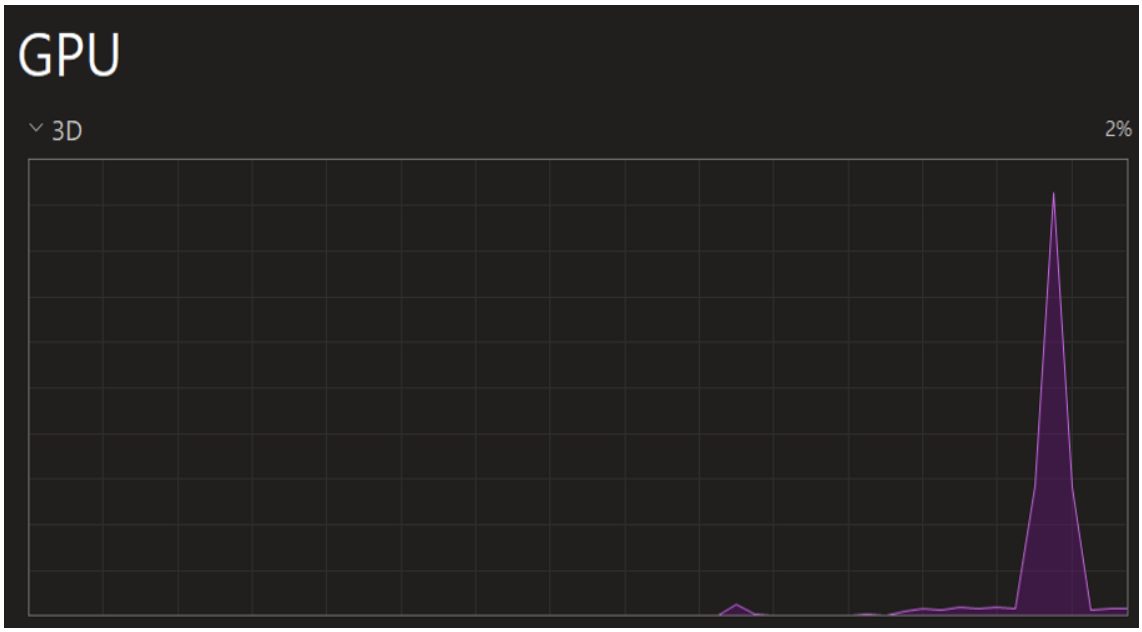

POST
http://localhost:3020/api/v1/predict/nnUNet
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00002-000-t1c.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 18.06 s
Size: 2.03 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"status": "success",
"message": "Predicción completada exitosamente",
"model": "nnUNet-V2",
"segmentation_file": null,
"output_path": "C:\\8282\\BraTS-GLI-00002-000-t1c_segmented_20260220_105340.nii.gz",
"elapsed_time": 17.181,
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",
"descripcion": "No visible tumor",
"pixeles": 8734951,
"porcentaje": 97.838

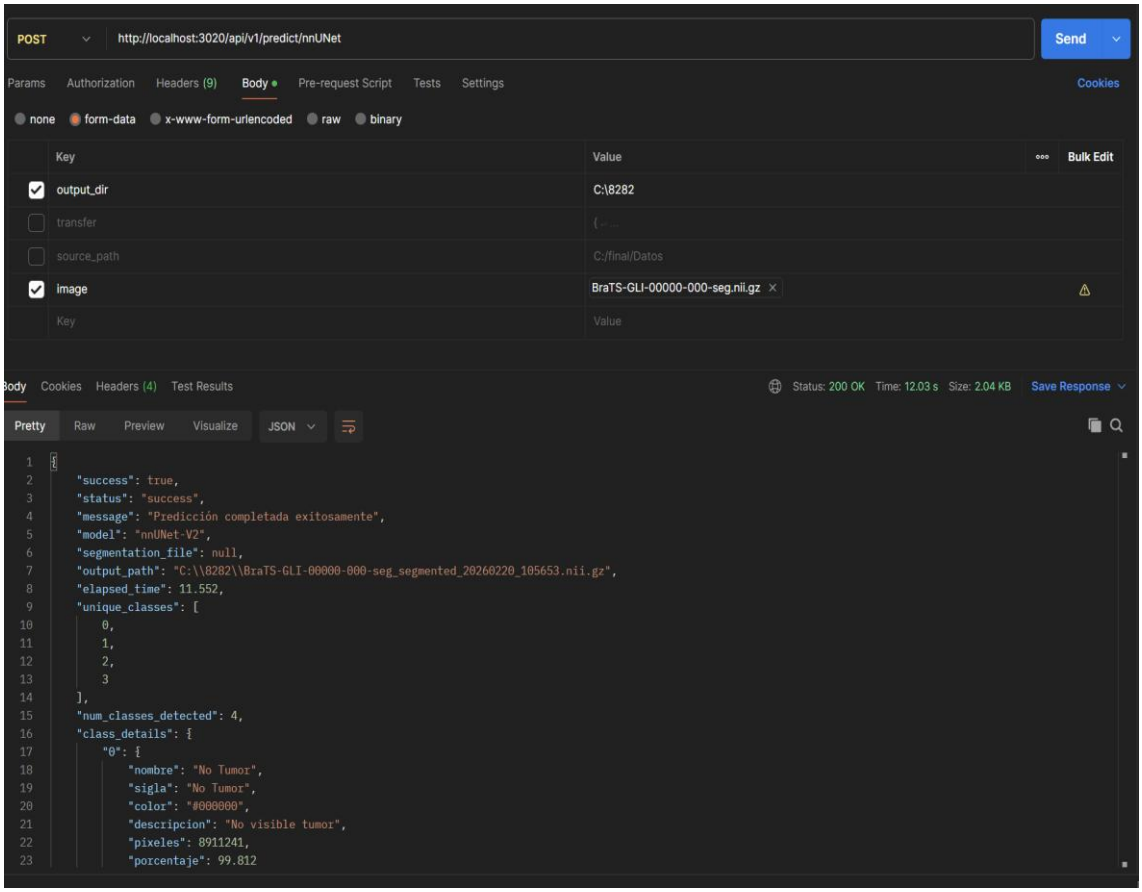
Prueba 6:


GPU
3D
43%

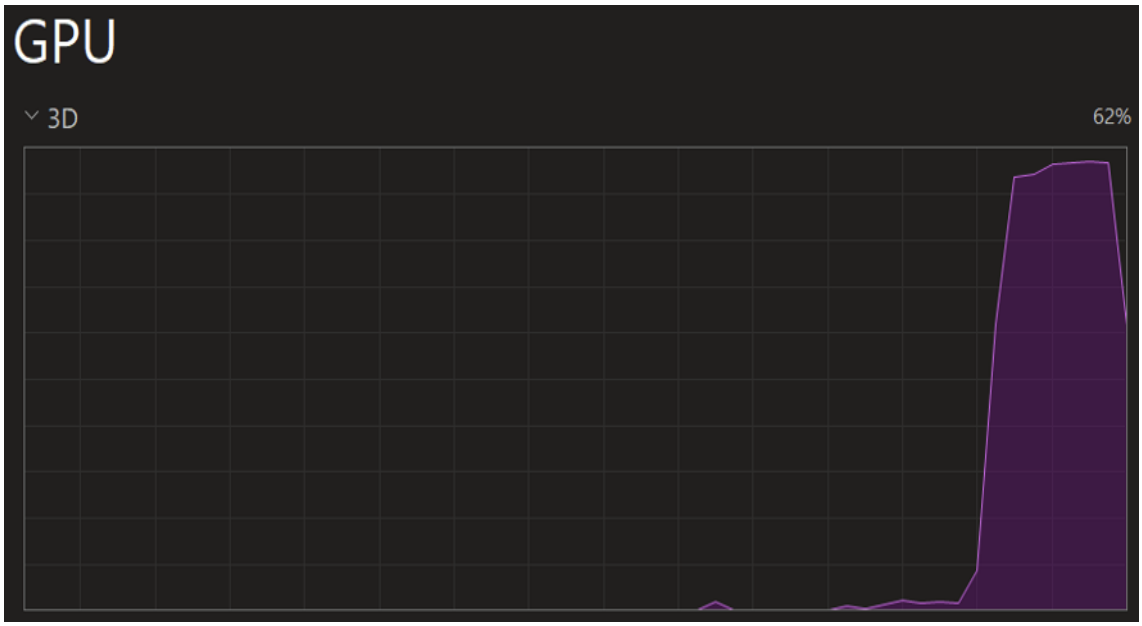

POST
http://localhost:3020/api/v1/predict/nnUNet
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00003-000-t1c.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 18.07 s
Size: 2.03 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"status": "success",
"message": "Predicción completada exitosamente",
"model": "nnUNet-V2",
"segmentation_file": null,
"output_path": "C:\\8282\\BraTS-GLI-00003-000-t1c_segmented_20260220_105505.nii.gz",
"elapsed_time": 17.412,
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",
"descripcion": "No visible tumor",
"pixeles": 8829293,
"porcentaje": 98.894

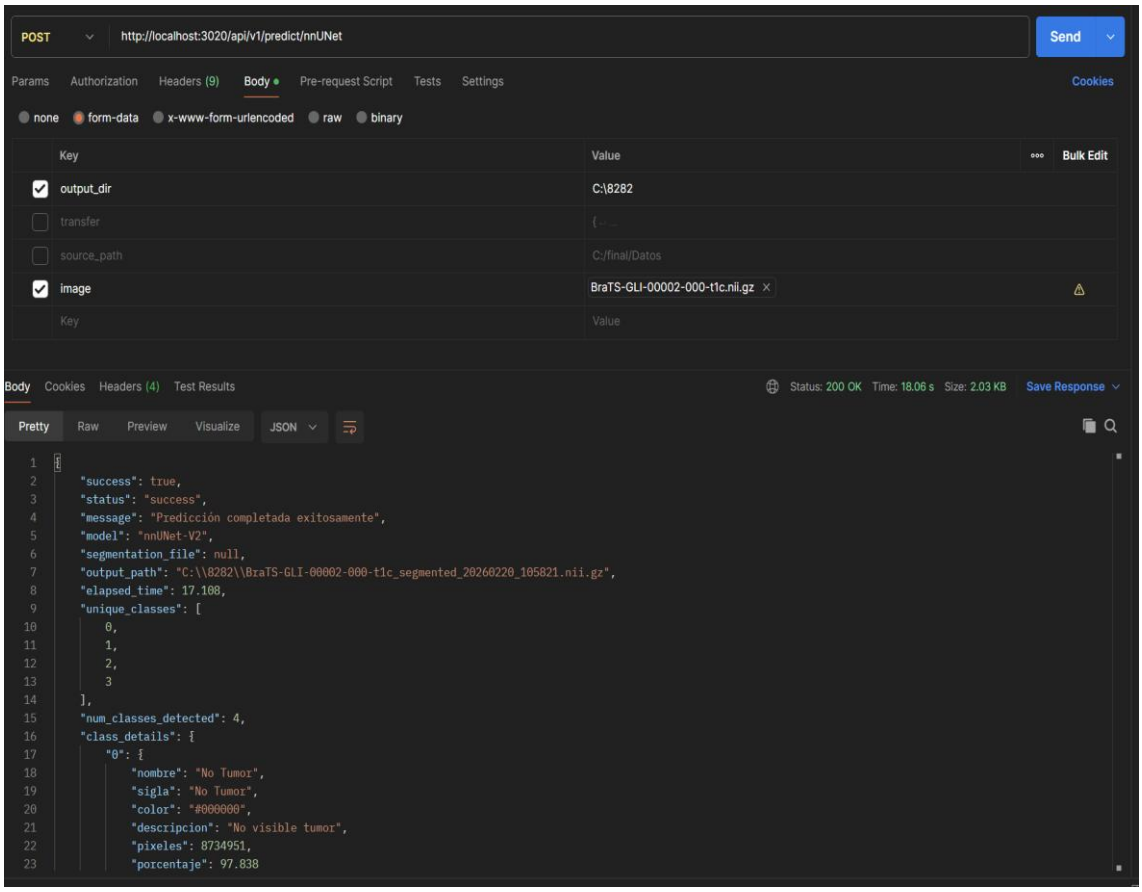
Prueba 7:


GPU
3D
2%

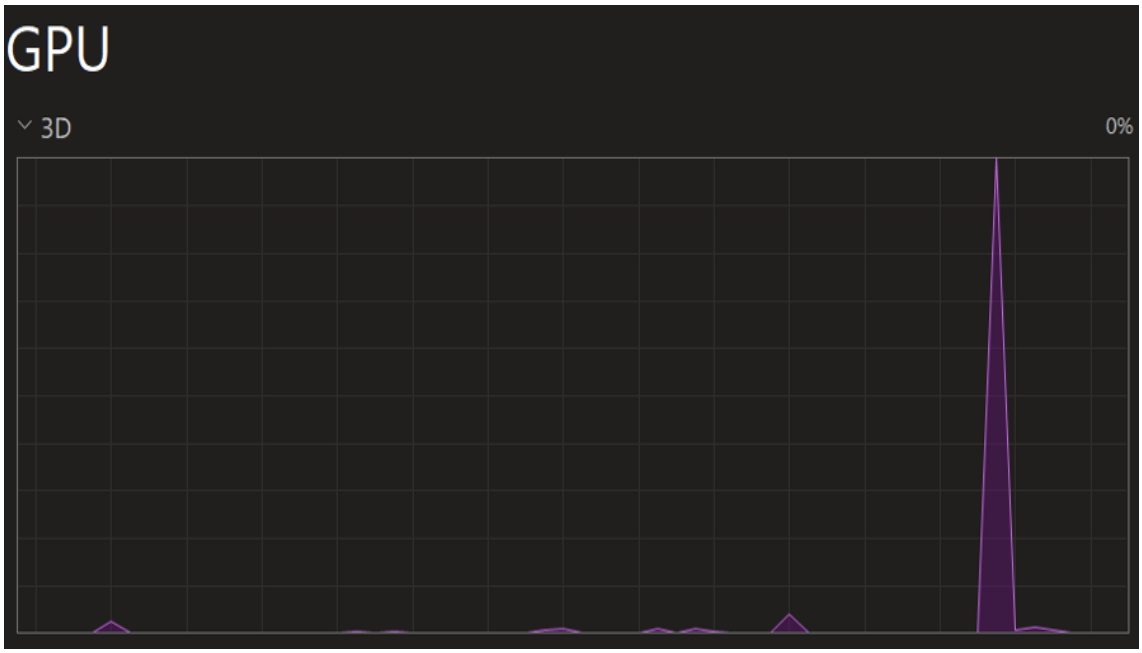

POST
http://localhost:3020/api/v1/predict/nnUNet
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
Image
BraTS-GLI-00000-000-seg.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 12.03 s
Size: 2.04 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"status": "success",
"message": "Predicción completada exitosamente",
"model": "nnUNet-V2",
"segmentation_file": null,
"output_path": "C:\\8282\\BraTS-GLI-00000-000-seg_segmented_20260220_105653.nii.gz",
"elapsed_time": 11.552,
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",
"descripcion": "No visible tumor",
"pixeles": 8911241,
"porcentaje": 99.812

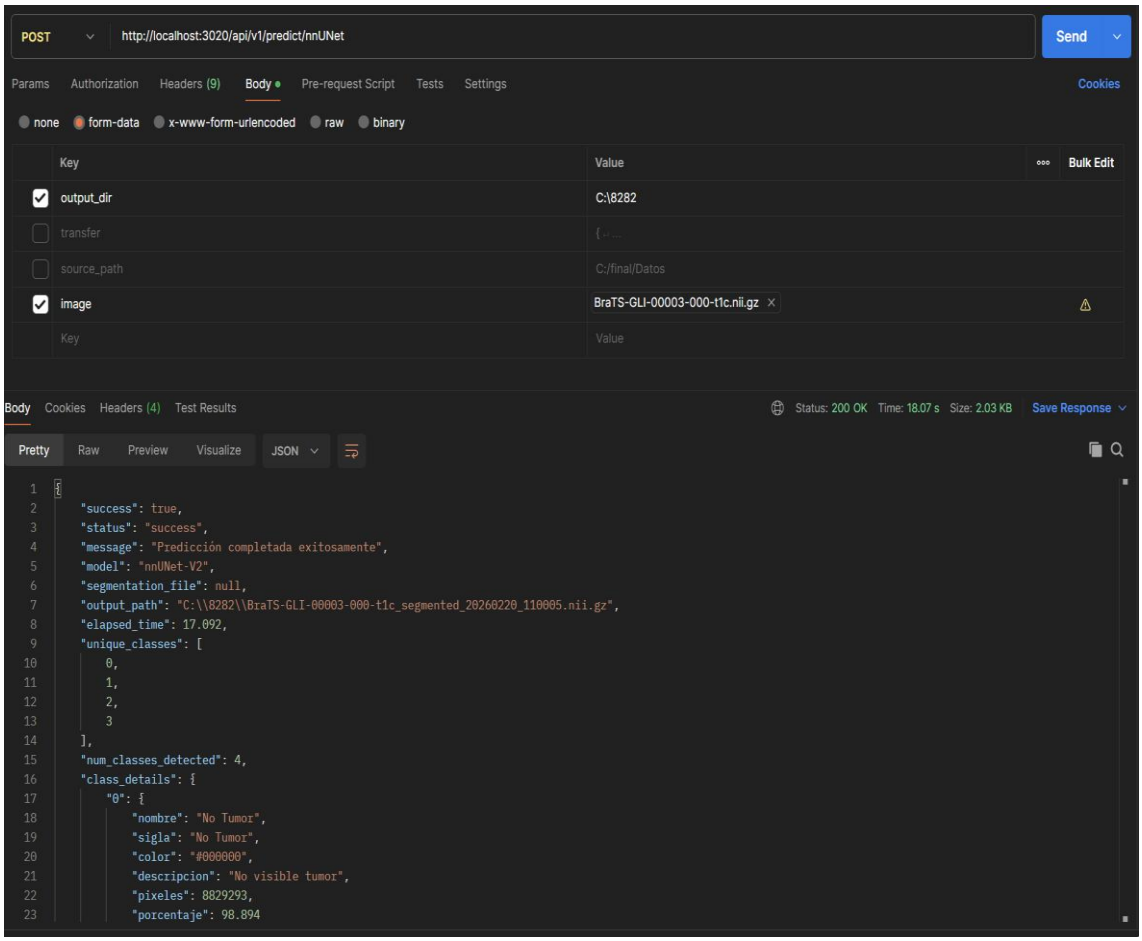
Prueba 8:


GPU
3D
62%

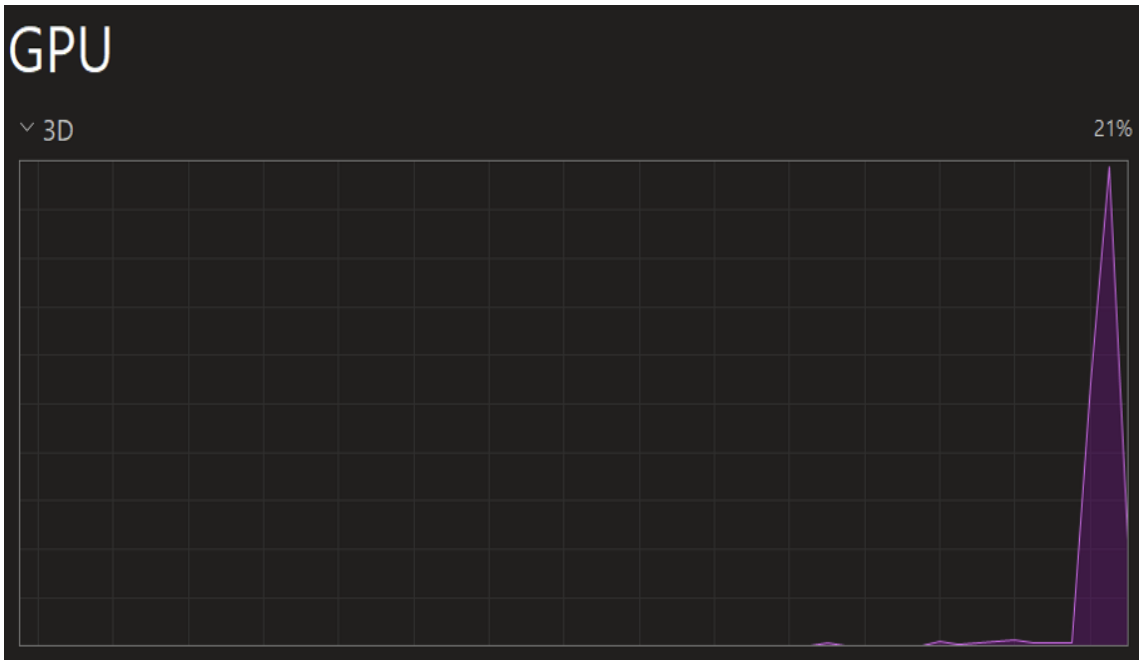

POST
http://localhost:3020/api/v1/predict/nnUNet
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00002-000-t1c.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 18.06 s
Size: 2.03 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"status": "success",
"message": "Predicción completada exitosamente",
"model": "nnUNet-V2",
"segmentation_file": null,
"output_path": "C:\\8282\\BraTS-GLI-00002-000-t1c_segmented_20260220_105821.nii.gz",
"elapsed_time": 17.108,
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",
"descripcion": "No visible tumor",
"pixeles": 8734951,
"porcentaje": 97.838

Prueba 9:


GPU
3D
0%


POST
http://localhost:3020/api/v1/predict/nnUNet
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00003-000-t1c.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 18.07 s
Size: 2.03 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"status": "success",
"message": "Predicción completada exitosamente",
"model": "nnUNet-V2",
"segmentation_file": null,
"output_path": "C:\\8282\\BraTS-GLI-00003-000-t1c_segmented_20260220_110005.nii.gz",
"elapsed_time": 17.092,
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",
"descripcion": "No visible tumor",
"pixeles": 8829293,
"porcentaje": 98.894

Prueba 10:


GPU
3D
21%

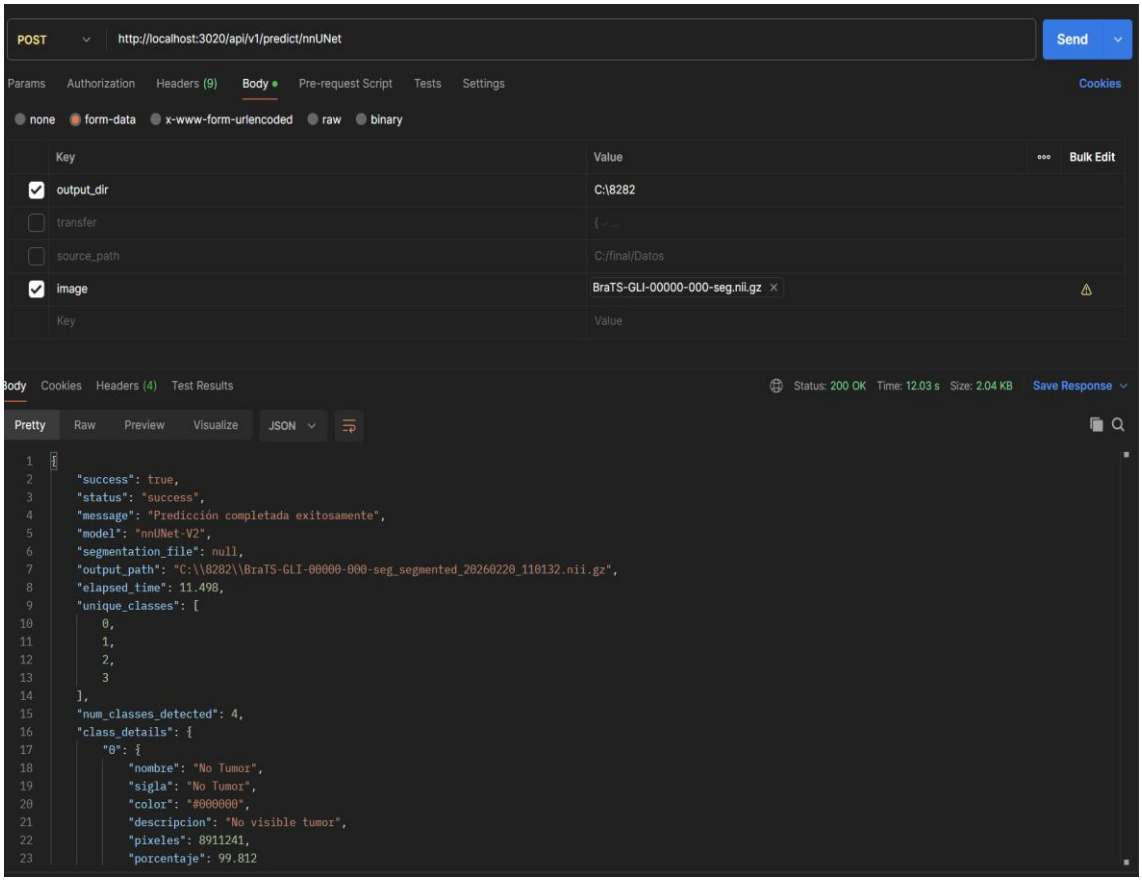

POST
http://localhost:3020/api/v1/predict/nnUNet
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
Key
Value
Bulk Edit
output_dir
C:\8282
transfer
source_path
C:/final/Datos
image
BraTS-GLI-00000-000-seg.nii.gz
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 12.03 s
Size: 2.04 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"success": true,
"status": "success",
"message": "Predicción completada exitosamente",
"model": "nnUNet-V2",
"segmentation_file": null,
"output_path": "C:\\8282\\BraTS-GLI-00000-000-seg_segmented_20260220_110132.nii.gz",
"elapsed_time": 11.498,
"unique_classes": [
0,
1,
2,
3
],
"num_classes_detected": 4,
"class_details": {
"0": {
"nombre": "No Tumor",
"sigla": "No Tumor",
"color": "#000000",
"descripcion": "No visible tumor",
"pixeles": 8911241,
"porcentaje": 99.812

Tabla 18

Resumen de métricas de desempeño del microservicio en entorno local (ASUS)

Nota. Datos obtenidos procesando imágenes con un peso promedio de 2.53 MB en un entorno Dockerizado. El uso de VRAM y GPU en el Modelo 1 se marca como N/D debido a su ejecución exclusiva en CPU.

Métrica Evaluada	Modelo 1 (UNet 3D MultiScale)	Modelo 2 (nnUNet)
Framework Base	TensorFlow	PyTorch
Arquitectura de Inferencia	CPU (Intel Core i9)	GPU (NVIDIA RTX 4060)
Parámetros del Modelo	~ 5.6 Millones	~ 88.2 Millones
Tamaño de Archivo del Modelo	64.7 MB	237 MB
Latencia Promedio por Imagen	1.66 s	15.66 s
Consumo Promedio de RAM	10.63 GB	N/D
Consumo Promedio de VRAM	N/D	0.65 GB
Uso de GPU durante inferencia	N/D	65.20 %

Tabla 19

Resumen de métricas de desempeño del microservicio en infraestructura de alta capacidad (NVIDIA DGX Spark)

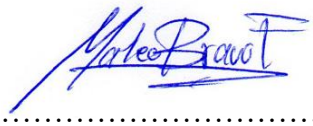
Nota. Las pruebas en el clúster DGX demuestran la escalabilidad de la arquitectura propuesta, reduciendo significativamente los tiempos de inferencia gracias a la memoria unificada y mayor capacidad de procesamiento en paralelo.

Métrica Evaluada	Modelo 1 (UNet 3D MultiScale)	Modelo 2 (nnUNet)
Arquitectura de Hardware	CPU Cortex-X925 / GPU Blackwell	CPU Cortex-X925 / GPU Blackwell
Memoria RAM Consumida	~ 42 % (Memoria Unificada)	~ 44.18 % (Memoria Unificada)
Uso de Procesador (CPU %)	6 %	14.99 %
Uso Gráfico (GPU %)	7 %	96 %
Latencia Promedio por Imagen	0.89 s	7.03 s

Universidad Católica de Cuenca	**AUTORIZACIÓN DE PUBLICACIÓN EN EL REPOSITORIO INSTITUCIONAL**

Mateo Josué Bravo Fernández portador de la cédula de ciudadanía Nº **0106758477**. En calidad de autor y titular de los derechos patrimoniales del trabajo de titulación **"IMPLEMENTACIÓN DE UN MICROSERVICIO ESCALABLE PARA EL ANÁLISIS AUTOMATIZADO DE CÁNCER CEREBRAL EN IMÁGENES DE RESONANCIA MAGNÉTICA MEDIANTE ARQUITECTURA DE CONTENEDORES"** de conformidad a lo establecido en el artículo 114 Código Orgánico de la Economía Social de los Conocimientos, Creatividad e Innovación, reconozco a favor de la Universidad Católica de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos y no comerciales. Autorizo además a la Universidad Católica de Cuenca**,** para que realice la publicación de éste trabajo de titulación en el Repositorio Institucional de conformidad a lo dispuesto en el artículo 144 de la Ley Orgánica de Educación Superior.

Cuenca, **08 de abril de 2026**



F: ……..

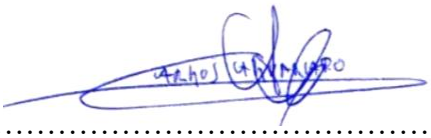
Mateo Josué Bravo Fernández

C.I. **0106758477**

Universidad Católica de Cuenca	**AUTORIZACIÓN DE PUBLICACIÓN EN EL REPOSITORIO INSTITUCIONAL**

Carlos Andrés Alvarado Gutiérrez portador de la cédula de ciudadanía Nº **0104134762**. En calidad de autor y titular de los derechos patrimoniales del trabajo de titulación **"IMPLEMENTACIÓN DE UN MICROSERVICIO ESCALABLE PARA EL ANÁLISIS AUTOMATIZADO DE CÁNCER CEREBRAL EN IMÁGENES DE RESONANCIA MAGNÉTICA MEDIANTE ARQUITECTURA DE CONTENEDORES"** de conformidad a lo establecido en el artículo 114 Código Orgánico de la Economía Social de los Conocimientos, Creatividad e Innovación, reconozco a favor de la Universidad Católica de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos y no comerciales. Autorizo además a la Universidad Católica de Cuenca**,** para que realice la publicación de éste trabajo de titulación en el Repositorio Institucional de conformidad a lo dispuesto en el artículo 144 de la Ley Orgánica de Educación Superior.

Cuenca, **08 de abril de 2026**



F: ……..

Carlos Andrés Alvarado Gutiérrez

C.I. **0104134762**