Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Modelo de interoperabilidad entre asistentes virtuales educativos heterogéneos

Tesis que presenta

Ing. Deneb Moliner Morales

para obtener el Grado de

Maestra en Ciencias

en Computación

Directores de Tesis

Dr. J. Guadalupe Rodríguez García
Dra. Sonia Guadalupe Mendoza Chapa

Ciudad de México **Agosto de 2026**

Resumen

En un panorama educativo cada vez más fragmentado, los asistentes virtuales especializados en dominios específicos —matemáticas, ciencias, idiomas, entre otros— operan de forma aislada, sin mecanismos estandarizados que permitan su comunicación, lo que incrementa la carga cognitiva del estudiante y limita el potencial de un ecosistema educativo verdaderamente integrado. Esta tesis propone un modelo de interoperabilidad de datos entre asistentes virtuales educativos heterogéneos, diseñado como un bus de datos común basado en el protocolo *Open Floor Protocol* (OFP) y en el patrón Adaptador (*Adapter*), que permite orquestar la delegación de turnos, la transferencia de contexto conversacional y el ensamblaje de respuestas entre asistentes especializados sin comprometer la soberanía de los datos de las instituciones educativas. El modelo se implementó como un prototipo funcional compuesto por un orquestador con clasificación semántica basada en *prompt engineering*, adaptadores que exponen a cada asistente virtual bajo un formato común y un mecanismo de registro dinámico en caliente que permite incorporar nuevos asistentes sin interrumpir el servicio ni modificar el núcleo de enrutamiento. Se implementó, además, un módulo de evaluación para medir de forma automatizada el desempeño individual y colectivo del ecosistema. La evaluación se desarrolló en dos fases: primero se caracterizó el desempeño aislado de diez asistentes especializados; después se evaluó el sistema orquestado completo utilizando cuatro modelos de lenguaje (Llama 3.1 8B, Gemma 2 9B, Mistral 12B y Qwen 2.5 14B) como cerebro del orquestador. Los resultados muestran que el principal cuello de botella del sistema no reside en la competencia individual de los asistentes especialistas, sino en la fase de ensamblaje de respuestas concurrentes; Qwen 2.5 14B fue la arquitectura con mejor desempeño global, aunque con mayor latencia. De forma complementaria, una evaluación de carga cognitiva con el cuestionario NASA-TLX ($n = 16$) mostró que el flujo orquestado reduce de forma estadísticamente significativa la demanda mental, el esfuerzo y la frustración percibidos por el usuario, sin alterar la tasa de acierto de la tarea. Estos hallazgos sugieren que el aporte principal del modelo propuesto no es mejorar la exactitud de la respuesta obtenida, sino reducir el costo cognitivo de coordinar múltiples asistentes heterogéneos, trasladando dicha carga del usuario hacia el sistema.

Palabras clave: interoperabilidad, asistentes virtuales educativos, *Open Floor Protocol*, sistemas multi-agente, carga cognitiva.

Abstract

In an increasingly fragmented educational landscape, virtual assistants specialized in specific domains—mathematics, science, languages, among others—operate in isolation, without standardized mechanisms that allow their communication. This increases the student's cognitive load and limits the potential of a truly integrated educational ecosystem. This thesis proposes a data interoperability model among heterogeneous educational virtual assistants, designed as a common data bus based on the Open Floor Protocol (OFP) and the Adapter pattern. This approach allows orchestrating turn delegation, conversational context transfer, and response assembly across specialized assistants without compromising the data sovereignty of educational institutions. The model was implemented as a functional prototype composed of an orchestrator featuring semantic classification based on prompt engineering, adapters that expose each virtual assistant under a common format, and a dynamic hot-registration mechanism that allows incorporating new assistants without interrupting the service or modifying the routing core. Furthermore, an evaluation module was implemented to automatically measure the individual and collective performance of the ecosystem. The evaluation was conducted in two phases: first, the isolated performance of ten specialized assistants was characterized; then, the fully orchestrated system was evaluated using four language models (Llama 3.1 8B, Gemma 2 9B, Mistral 12B, and Qwen 2.5 14B) as the orchestrator's brain. The results show that the main bottleneck of the system does not lie in the individual competence of the specialist assistants, but rather in the concurrent response assembly phase; Qwen 2.5 14B was the architecture with the best overall performance, albeit with higher latency. Complementarily, a cognitive load evaluation using the NASA-TLX questionnaire ($n = 16$) showed that the orchestrated flow statistically significantly reduces the mental demand, effort, and frustration perceived by the user, without altering the task success rate. These findings suggest that the main contribution of the proposed model is not improving the accuracy of the obtained response, but reducing the cognitive cost of coordinating multiple heterogeneous assistants, shifting this burden from the user to the system.

Keywords: interoperability, educational virtual assistants, Open Floor Protocol, multi-agent systems, cognitive load.

Agradecimientos

El camino hacia la culminación de este proyecto no se recorrió en solitario. Este logro es el resultado del apoyo, la paciencia y el aliento de muchas personas e instituciones que creyeron en mí. Aunque las palabras nunca serán suficientes para expresar mi gratitud, deseo dedicarles este espacio con el más sincero agradecimiento.

A mi familia, por su apoyo incondicional, por acompañarme en los momentos más difíciles y por motivarme a seguir adelante. Gracias por su cariño, su paciencia y por estar siempre a mi lado.

A mis amigos en la distancia, aquellos que la vida puso en mi camino desde el círculo infantil, durante mi formación como ingeniera y a lo largo de mi trayectoria profesional. Aunque hoy nos separen kilómetros y fronteras, cada mensaje y cada videollamada fueron un recordatorio de que nunca estuve sola. Gracias por mantener viva nuestra amistad y por celebrar conmigo este logro desde distintos rincones del mundo.

De manera muy especial, a Miguel Alessandro, a quien tuve la fortuna de conocer al iniciar esta etapa de maestría. Gracias por tu amistad, por escucharme en los momentos de incertidumbre y por hacer de este proceso una experiencia más llevadera.

A la Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECIHTI), al Centro de Investigación y de Estudios Avanzados (CINVESTAV) y al Departamento de Computación, por brindarme el respaldo institucional y el entorno académico necesarios para desarrollar esta investigación.

A mis directores de tesis, el Dr. José Guadalupe Rodríguez García y la Dra. Sonia Guadalupe Mendoza Chapa, por su guía, dedicación y valiosa retroalimentación durante este proyecto. Agradezco el rigor académico y la confianza con la que orientaron este trabajo.

A los miembros del jurado, por el tiempo dedicado a la revisión de esta tesis y por las observaciones que enriquecieron el resultado final.

A mis compañeros del laboratorio, por hacer del espacio de trabajo un ambiente de colaboración y compañerismo. Gracias por las conversaciones y los consejos que hicieron este camino más ameno.

Finalmente, agradezco a todas las personas que, de una u otra manera, compartieron sus conocimientos o me brindaron una palabra de aliento. A todos ustedes, gracias por formar parte de este logro.

Abreviaturas

A2A *Agent to Agent* (Protocolo Agente a Agente)

ACL *Agent Communication Language* (Lenguaje de Comunicación de Agentes)

ACP *Agent Communication Protocol* (Protocolo de Comunicación de Agentes)

AMQP *Advanced Message Queuing Protocol*

ANP *Agent Network Protocol* (Protocolo de Red de Agentes)

API *Application Programming Interface* (Interfaz de Programación de Aplicaciones)

CNP *Contract Net Protocol* (Protocolo de Red de Contratos)

CPU *Central Processing Unit* (Unidad Central de Procesamiento)

FIPA *Foundation for Intelligent Physical Agents* (Fundación para Agentes Físicos Inteligentes)

GPU *Graphics Processing Unit* (Unidad de Procesamiento Gráfico)

IA Inteligencia Artificial

JADE *Java Agent Development Framework* (Marco de Trabajo para el Desarrollo de Agentes en Java)

LLM *Large Language Model* (Modelo de Lenguaje de Gran Escala)

MCP *Model Context Protocol* (Protocolo de Contexto del Modelo)

NASA-TLX *NASA Task Load Index* (Índice de Carga de Tareas de la NASA)

NLU *Natural Language Understanding* (Comprensión del Lenguaje Natural)

OFP *Open Floor Protocol* (Protocolo de Piso Abierto)

ONU Organización de las Naciones Unidas

RAG *Retrieval-Augmented Generation* (Generación Aumentada por Recuperación)

RAM *Random Access Memory* (Memoria de Acceso Aleatorio)

ROM *Read-Only Memory* (Memoria de Solo Lectura)

RTLX *Raw TLX* (Variante del NASA-TLX)

SDK *Software Development Kit* (Kit de Desarrollo de Software)

SOLID Principios de Diseño Orientado a Objetos (Single responsibility, Open-closed, Liskov substitution, Interface segregation, Dependency inversion)

UML *Unified Modeling Language* (Lenguaje Unificado de Modelado)

URI *Uniform Resource Identifier* (Identificador de Recursos Uniforme)

UUID *Universally Unique Identifier* (Identificador Único Universal)

VRAM *Video Random Access Memory* (Memoria de Acceso Aleatorio de Video)

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En un panorama educativo cada vez más fragmentado, los asistentes virtuales se han consolidado como herramientas clave para el aprendizaje personalizado y la retroalimentación inmediata, especialmente en entornos digitales y a distancia [1, 2]. Sin embargo, su proliferación ha generado una dispersión significativa: los estudiantes y docentes deben interactuar con múltiples plataformas y sistemas que no comparten datos ni contexto de forma fluida, lo que dificulta la continuidad pedagógica y aumenta la carga cognitiva [2]. Esta fragmentación se agrava en contextos multidisciplinarios, donde un asistente especializado en Matemáticas no puede integrarse fácilmente con uno enfocado en Física o en otras disciplinas, limitando el potencial de un ecosistema educativo unificado y colaborativo.

Este proyecto aborda precisamente ese reto: la integración de asistentes virtuales educativos heterogéneos, mediante un modelo que facilite el intercambio controlado de datos y la evaluación colaborativa en tiempo real. En el ámbito de la ingeniería de software, un modelo se define como una representación abstracta de un sistema bajo estudio que captura sus características estructurales y de comportamiento más significativas, omitiendo aquellos detalles que no resultan relevantes para su diseño, análisis y comunicación [3, 4]. Inspirado en los avances recientes en protocolos de comunicación para sistemas compuestos por asistentes conversacionales, el enfoque se centra en el ámbito educativo para superar barreras técnicas clave, como la incompatibilidad de protocolos y la ausencia de métricas integradas de rendimiento [5, 6].

El modelo propuesto busca facilitar una interacción más coherente entre asistentes virtuales educativos heterogéneos, promoviendo la continuidad del proceso de aprendizaje y reduciendo las barreras que surgen de su aislamiento actual. De esta forma, se pretende contribuir a un ecosistema de IA educativa más integrado, donde los asistentes virtuales puedan colaborar de manera efectiva para apoyar al estudiante en contextos diversos.

Más allá de los aspectos técnicos, esta investigación se sitúa en la intersección entre el diseño de asistentes virtuales educativos interoperables y las arquitecturas de coordinación basadas en agentes y la facilitación de la interacción humano-computadora, promoviendo un uso responsable y pedagógicamente orientado de la IA en la educación [7, 8]. Los resultados esperados incluyen un prototipo funcional que demuestre la via-

bilidad técnica de la interoperabilidad propuesta, acompañado de una evaluación que combine indicadores técnicos y consideraciones pedagógicas para valorar su potencial aporte al campo.

1.1 Motivación

El surgimiento y la rápida expansión de los asistentes virtuales han transformado la interacción con la tecnología y la accesibilidad a la información. Inicialmente populares en dispositivos de consumo masivo (e.g., Siri [9], Alexa [10], Asistente de Google [11]), su adopción se ha extendido rápidamente a sectores especializados. Particularmente en el ámbito educativo, estas herramientas actúan como tutores virtuales, facilitadores administrativos o sistemas de apoyo al aprendizaje [12], proporcionando una interacción conversacional capaz de ofrecer retroalimentación inmediata y adaptada al ritmo de cada alumno [13].

Este crecimiento tecnológico experimentó una aceleración sin precedentes durante la crisis sanitaria global de 2020. El cierre de las aulas obligó a una transición masiva hacia modalidades digitales, poniendo en evidencia el potencial de estas tecnologías para brindar apoyo continuo en momentos donde la atención docente directa no estaba disponible [14]. Aún frente a las brechas de infraestructura y conectividad, el desarrollo de herramientas educativas basadas en IA creció significativamente, impulsado por la búsqueda de soluciones inclusivas [15].

Sin embargo, este rápido crecimiento ha generado una gran diversidad de sistemas, lo que se refleja en dos tendencias clave que resultan problemáticas:

1. **Especialización:** existen numerosos asistentes virtuales diseñados para propósitos muy específicos (e.g., matemáticas, salud e idiomas) optimizados para su sector, pero operando de forma aislada e ignorando el contexto global del estudiante.
2. **Fragmentación:** el usuario moderno asume múltiples roles, lo que lo obliga a interactuar con diversas interfaces inconexas, cada una con su propia tecnología y limitaciones, resultando en un aprendizaje fragmentado y poco eficiente.

Actualmente, el ecosistema digital educativo opera bajo un paradigma de islas digitales. Un estudiante puede utilizar una herramienta para razonar lógicamente un problema y otra distinta para redactar un reporte; sin embargo, al no existir comunicación entre ellas, el aprendizaje no se transfiere. Este aislamiento trasciende la simple molestia técnica y tiene un impacto directo en el proceso de aprendizaje: el aumento de la carga cognitiva [16]. Cuando un estudiante debe alternar entre múltiples interfaces que no se comunican, su cerebro agota valiosos recursos tratando de gestionar la tecnología, desviando la atención del aprendizaje profundo.

La presente tesis de maestría se motiva precisamente en la necesidad de superar esta barrera arquitectónica. Se propone el diseño de un modelo de interoperabilidad que actúe como un bus de datos común y un gestor de servicios, permitiendo que asistentes virtuales heterogéneos se comuniquen, se coordinen y cooperen de manera efectiva. Al lograr que los asistentes colaboren y compartan información de forma transparente bajo un protocolo estandarizado, se elimina el ruido operativo y se

reduce la carga cognitiva del alumno. Asimismo, este enfoque orquestado permite integrar un mecanismo de métricas para evaluar con precisión el desempeño de la colaboración entre asistentes, mejorando la eficiencia pedagógica.

Aporte social y democratización educativa

Más allá de la innovación arquitectónica, esta investigación persigue un impacto social tangible alineado con el Objetivo de Desarrollo Sostenible 4: garantizar una educación inclusiva, equitativa y de calidad [17].

En regiones como América Latina, la integración de la IA en el aula se enfrenta a la barrera histórica de la brecha digital. La dependencia de conectividad de banda ancha y de servicios comerciales en la nube excluye a estudiantes en contextos vulnerables o zonas rurales, exacerbando las desigualdades estructurales preexistentes [18]. El modelo propuesto mitiga esta exclusión al sentar las bases para una orquestación descentralizada de IA. Al permitir que múltiples asistentes operen, colaboren y procesen la información de manera local directamente en la infraestructura de la institución, se independiza el proceso de aprendizaje del acceso ininterrumpido a Internet.

Asimismo, esta arquitectura descentralizada fomenta la soberanía tecnológica y protege la privacidad de los usuarios. La centralización masiva de datos educativos en servidores extranjeros plantea serios riesgos éticos [15]. Al procesar las consultas de los estudiantes mediante modelos de lenguaje locales y evitar que la información sea extraída por corporaciones de terceros, el ecosistema garantiza un entorno digital seguro, en estricto cumplimiento con las directrices internacionales para el uso de IA en menores [19]. En este sentido, el aporte social de la presente tesis radica en demostrar que es tecnológicamente viable construir un ecosistema educativo avanzado y colaborativo que sea accesible, privado y soberano frente a las limitaciones de infraestructura de la región.

1.2 Antecedentes

La presente investigación se fundamenta directamente en el trabajo de tesis de maestría de Avalos Montiel (2023), que abordó la asistencia virtual desde una perspectiva de interacción y conciencia del contexto [20]. Este trabajo se convierte en el punto de partida y la principal justificación para el salto arquitectónico propuesto.

Esta tesis constituyó un avance significativo al diseñar un modelo de navegación web por voz, basado en contexto. Sus principales contribuciones se orientaron a mejorar la accesibilidad y la efectividad de la búsqueda al integrar el análisis del perfil del usuario y el contenido web. El resultado fue la consolidación de un sistema de asistente virtual funcional, enfocado en una tarea específica, la navegación web asistida por voz. Este logro demostró la viabilidad de un asistente virtual que no solo interpreta comandos de voz, sino que también reacciona de manera contextual a las necesidades del usuario.

La principal limitación inherente al trabajo de Avalos Montiel [20] es que el modelo

fue concebido como un sistema aislado. Si bien resolvió eficazmente la interacción y la búsqueda en su entorno específico, no contempla la interoperabilidad con otros asistentes virtuales o servicios especializados. Este aislamiento sistémico impide que el asistente pueda delegar tareas complejas o colaborar con otros asistentes virtuales optimizados para distintas funciones.

La presente tesis de maestría se basa en el enfoque contextual del trabajo [20] y se centra en superar esta limitación. En lugar de rediseñar la capa de interacción, se propone desarrollar un modelo de interoperabilidad que permita la comunicación, coordinación y cooperación entre asistentes virtuales heterogéneos. Este enfoque aborda la fragmentación de la asistencia virtual en entornos educativos.

1.3 Planteamiento del problema

En la educación y el aprendizaje digital, los asistentes virtuales basados en IA han emergido como herramientas fundamentales para potenciar la personalización del aprendizaje, el soporte pedagógico y la interacción entre estudiantes y contenidos educativos. Estos sistemas facilitan el acceso a recursos adaptativos, fomentan el aprendizaje autorregulado y mejoran la eficiencia en entornos educativos diversificados [1, 2]. Sin embargo, la proliferación de asistentes virtuales educativos heterogéneos desarrollados con arquitecturas, protocolos y tecnologías disímiles plantea desafíos significativos en términos de interoperabilidad y evaluación colaborativa, lo que limita su integración efectiva en ecosistemas educativos complejos.

El problema central radica en la fragmentación de estos sistemas heterogéneos, que operan de manera aislada y no disponen de mecanismos estandarizados para comunicarse e interactuar entre sí. Esta heterogeneidad impide la colaboración fluida entre asistentes, resultando en evaluaciones pedagógicas ineficientes que no aprovechan el potencial colectivo de múltiples asistentes para enriquecer el proceso de enseñanza-aprendizaje [2, 21]. Por ejemplo, en escenarios donde coexisten asistentes especializados en diferentes dominios (como tutoría en operaciones matemáticas, ciencias o idiomas), la ausencia de interoperabilidad genera redundancias, inconsistencias en la retroalimentación y una evaluación fragmentada del rendimiento estudiantil, afectando la calidad del aprendizaje personalizado y colaborativo.

Esta problemática se agrava en contextos educativos globales y multidisciplinarios, donde la diversidad de sistemas de IA —provenientes de proveedores distintos o basados en modelos variados— no solo complica la integración técnica, también plantea barreras pedagógicas y éticas, como la inequidad en el acceso a evaluaciones integradas y la falta de transparencia en los procesos de interacción [6, 7]. Aunque existen avances en protocolos de comunicación para asistentes de IA, su aplicación se concentra principalmente en dominios no educativos, dejando un vacío en modelos específicos para la interoperabilidad y la evaluación colaborativa en entornos de aprendizaje heterogéneos [5].

Estas limitaciones subrayan la necesidad de desarrollar un modelo de interoperabilidad que aborde la comunicación entre asistentes virtuales educativos heterogéneos y facilite su evaluación colaborativa, contribuyendo así a superar las barreras identifi-

cadas y a potenciar la eficacia pedagógica en la educación contemporánea.

1.4 Hipótesis

Un modelo diseñado para promover la interoperabilidad entre asistentes virtuales educativos heterogéneos facilitará una colaboración más efectiva, lo que resultará en una mejora significativa de la calidad del aprendizaje personalizado y la eficiencia pedagógica en entornos educativos diversificados.

1.5 Objetivos

Objetivo general

Desarrollar un modelo de interoperabilidad que permita el intercambio, la integración y el uso de datos entre asistentes virtuales heterogéneos enfocados a la educación.

Objetivos específicos

1. Identificar los requerimientos de interoperabilidad de datos entre asistentes virtuales educativos heterogéneos, analizando sus protocolos y limitaciones de comunicación.
2. Diseñar un modelo para estandarizar el intercambio de datos y la delegación de turnos entre asistentes virtuales.
3. Desarrollar un orquestador que coordine la colaboración entre asistentes virtuales heterogéneos, manejando descubrimiento, delegación y coherencia conversacional.
4. Implementar un prototipo del modelo, integrando varios asistentes virtuales educativos heterogéneos.
5. Definir e integrar un módulo de métricas para evaluar el desempeño individual y en conjunto de los asistentes virtuales.
6. Evaluar la interoperabilidad, el mantenimiento del contexto educativo y la carga cognitiva percibida por el usuario final al interactuar con el modelo propuesto utilizando el cuestionario NASA-TLX.

1.6 Organización del documento

El presente trabajo de tesis se organiza de la siguiente manera:
En el Capítulo 2 se presenta el estado del arte y los conceptos fundamentales relacionados con la investigación. Se revisan los principales trabajos relacionados y avances en interoperabilidad de asistentes virtuales educativos, protocolos de comunicación multi-agente y estándares abiertos como el protoclo OFP. Asimismo, se definen los conceptos clave que sustentan el modelo propuesto, tales como asistentes virtua-

les heterogéneos, interoperabilidad conversacional, gestión del piso conversacional y evaluación colaborativa en entornos educativos.

En el Capítulo 3 se detalla el análisis y diseño del modelo de interoperabilidad. Se describen los requerimientos funcionales y no funcionales, la arquitectura general del sistema (incluyendo el componente coordinador central, los adaptadores para cada asistente y el entorno de ejecución), los diagramas de diseño (flujo de mensajes, componentes y secuencia) y la integración del protocolo OFP como base de comunicación estandarizada.

El Capítulo 4 describe la implementación del prototipo. Se explica la configuración del entorno con Docker, el desarrollo del componente coordinador, la construcción de los adaptadores para los asistentes seleccionados, la estandarización de mensajes según OFP y las decisiones técnicas adoptadas para garantizar seguridad, portabilidad y mantenimiento de contexto conversacional.

En el Capítulo 5 se expone la metodología de evaluación y se analizan los resultados obtenidos tras la implementación del modelo. El análisis se divide en dos fases: primero, la validación del rendimiento individual de los diez asistentes virtuales especialistas de forma aislada; segundo, la evaluación del componente orquestador utilizando cuatro modelos de lenguaje de gran escala: Llama, Gemma, Mistral y Qwen. Se detallan las métricas empleadas: precisión, exhaustividad y F1-Score, similitud semántica y latencia global. Adicionalmente, se incluye una evaluación experimental de la carga cognitiva percibida por los usuarios mediante el cuestionario NASA-TLX (en su variante RTLX), contrastando el esfuerzo demandado por un flujo de interacción manual frente al ecosistema orquestado.

Finalmente, en el Capítulo 6 se presentan las conclusiones de la investigación y se verifica el cumplimiento de la hipótesis y de los objetivos específicos. Además, se enumeran las aportaciones del trabajo, las limitaciones del modelo propuesto y las posibles líneas de trabajo futuro.

Capítulo 2

Estado del arte

En este capítulo se exponen los fundamentos conceptuales y tecnológicos necesarios para comprender el modelo de interoperabilidad propuesto, así como una revisión del estado del arte en las áreas relacionadas. Primeramente se presenta el marco teórico sobre asistentes virtuales educativos (cf. Sección 2.1), seguida de una inmersión en los sistemas multi-agente contemporáneos (cf. Sección 2.2) y los protocolos estandarizados de comunicación (cf. Sección 2.3). Posteriormente, se analizan los trabajos relacionados más relevantes en el ámbito educativo y de despliegue de IA (cf. Sección 2.4). Finalmente, se sintetizan las limitaciones de las soluciones actuales para identificar la brecha de investigación que justifica la arquitectura agnóstica planteada en este trabajo (cf. Sección 2.5).

2.1 Asistentes virtuales en la educación

En los últimos años ha habido un rápido desarrollo en el campo de los asistentes virtuales educativos [1], los cuales están diseñados para interactuar con los estudiantes a través de interfaces conversacionales, apoyando la personalización del aprendizaje y la retroalimentación inmediata [2]. Este avance significativo ha remodelado intrínsecamente la forma en que se interactúa con el conocimiento educativo, pasando de sistemas aislados a ecosistemas colaborativos potenciales, aunque aún limitados por la heterogeneidad técnica [6].

Un asistente virtual se define como un sistema de software inteligente capaz de interactuar con los usuarios mediante lenguaje natural —ya sea texto, voz o de manera multimodal— con el propósito de proporcionar información, ejecutar tareas o asistir procesos cognitivos de forma autónoma o semi-autónoma [22]. Estos sistemas integran técnicas de procesamiento del lenguaje natural, gestión del diálogo y aprendizaje automático para simular interacciones humanas orientadas a objetivos específicos. Es importante precisar la relación conceptual entre los términos asistente y agente, dado que a lo largo de este capítulo ambos términos aparecerán con frecuencia y no son sinónimos. Un asistente [22] se caracteriza por su rol de interfaz: es la entidad con la que el usuario interactúa directamente mediante lenguaje natural para obtener información o ejecutar tareas. Un agente, en cambio, corresponde a un concepto más

amplio proveniente de los sistemas distribuidos de IA: una entidad computacional capaz de percibir su entorno, razonar y actuar de forma autónoma, ya sea de manera individual o en coordinación con otras entidades equivalentes dentro de un sistema multi-agente [23], sin que medie necesariamente una interacción conversacional directa con un humano. Este concepto incluye a los agentes basados en LLM, entendidos como entidades capaces de percibir, razonar y actuar sobre su entorno mediante las capacidades del propio LLM [24]. Así, un mismo componente de software puede describirse como "asistente" cuando se enfatiza su papel de interfaz frente al estudiante y como "agente" cuando se analiza su papel dentro de la arquitectura de coordinación, delegación o razonamiento del sistema multi-agente subyacente. En esta tesis se utilizará el término asistente virtual cuando se describa la interacción con estudiantes y docentes, mientras que el término agente se reservará para describir los componentes internos responsables del razonamiento, la coordinación y la comunicación dentro de la arquitectura propuesta.

En el ámbito educativo, los asistentes virtuales actúan como tutores digitales, orientadores académicos o asistentes de soporte institucional, facilitando experiencias de aprendizaje personalizadas, retroalimentación inmediata y disponibilidad continua. Diversos estudios han evidenciado que su incorporación puede mejorar la participación estudiantil, la retención de contenidos y el acceso a servicios académicos, especialmente en entornos virtuales o híbridos [1]. No obstante, la rápida evolución tecnológica de estos sistemas ha propiciado la coexistencia de múltiples enfoques de diseño y plataformas heterogéneas, lo que introduce desafíos de mantenimiento, integración y escalabilidad dentro de los ecosistemas educativos [25].

2.1.1 Evolución arquitectónica y topológica de los asistentes virtuales

Para fundamentar el diseño de entornos educativos inteligentes, es necesario clasificar los asistentes virtuales no solo por su capacidad de procesamiento lingüístico, sino también por la topología de su arquitectura de software [22]. Esta aproximación permite distinguir sus capacidades de adaptación, sus limitaciones operacionales y la transición desde componentes aislados hacia sistemas distribuidos:

- **Asistentes basados en reglas:** corresponden a implementaciones deterministas sustentadas en árboles de decisión o estructuras lógicas de tipo *if–then*. Fueron ampliamente utilizados en los primeros chatbots educativos debido a su simplicidad y controlabilidad, resultando adecuados para la resolución de preguntas frecuentes o flujos administrativos predefinidos [26]. Sin embargo, su rigidez intrínseca impide la adaptación a diálogos abiertos y limita drásticamente su escalabilidad en entornos de aprendizaje dinámicos.
- **Asistentes basados en NLU:** incorporan técnicas de aprendizaje automático clásico para segmentar la interacción mediante la clasificación de intenciones, el reconocimiento de entidades y la gestión de diálogos modulares basados en estados [27, 28]. *Frameworks* de desarrollo como Rasa o Dialogflow permiten mapear las entradas lingüísticas del usuario hacia acciones estructuradas predefinidas [29]. Aunque superan la rigidez sintáctica de las reglas, siguen restringidos

a dominios cerrados y tareas transaccionales, requiriendo un entrenamiento manual exhaustivo para cada nueva intención lingüística.

- **Asistentes generativos basados en LLM:** son sistemas impulsados por modelos de lenguaje de gran escala fundamentados en la arquitectura *Transformer* y sus mecanismos de atención contextual [30]. A diferencia de los enfoques basados en NLU, prescinden de la definición estricta de intenciones, permitiendo la generación de respuestas abiertas, coherentes y contextualmente adaptativas. En el contexto de arquitecturas soberanas y de ejecución local, el surgimiento de familias de modelos de código abierto eficientes (como LLaMA) ha viabilizado su despliegue en entornos locales de menor coste computacional [31]. Esto transforma al asistente en un componente agnóstico a servicios en la nube, capaz de realizar razonamiento contextualizado y apoyo pedagógico flexible sin comprometer la privacidad de los datos del estudiante.
- **Componentes analíticos y de soporte multimodal:** representan módulos especializados que expanden las capacidades de interacción y diagnóstico del sistema. Por un lado, la integración de canales multimodales (voz y visión por computadora, texto, gestos) enriquece la accesibilidad en escenarios de aprendizaje ubicuo [22]. Por otro lado, los servicios de soporte analítico ejecutan tareas en el *backend*, como el análisis de sentimientos o la detección de patrones conductuales [32]. Estos componentes no operan de forma aislada, sino como herramientas utilitarias que asisten la toma de decisiones del entorno conceptual.
- **Sistemas multi-agente colaborativos:** consisten en entornos distribuidos donde múltiples agentes autónomos y especializados cooperan entre sí para resolver tareas pedagógicas complejas que superan las capacidades de cualquier modelo monolítico [23]. El valor fundamental de esta topología en la educación radica en su capacidad para gestionar la heterogeneidad; un único ecosistema puede integrar agentes basados en reglas (para trámites), asistentes NLU (para evaluaciones estructuradas) y agentes con LLM locales (para tutorías abiertas). Para que esta colaboración sea viable, el sistema requiere de un componente orquestador y del empleo de protocolos de comunicación estandarizados que resuelvan la interoperabilidad sintáctica y semántica entre los nodos, garantizando un entorno modular y escalable.

La Figura 2.1 ilustra cómo las arquitecturas de agentes individuales, con diferentes paradigmas de procesamiento, se integran y encapsulan dentro de una topología multi-agente coordinada.

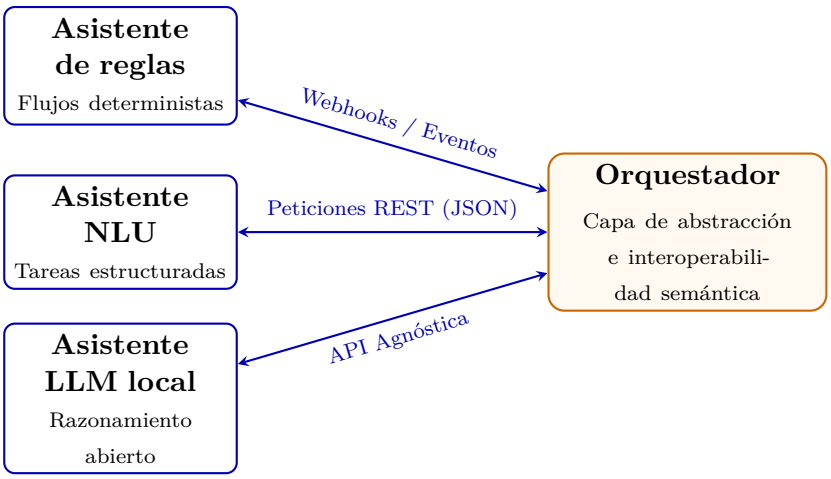


Figura 2.1: Ecosistema educativo basado en la orquestación de asistentes heterogéneos.

2.1.2 Arquitecturas de despliegue para asistentes virtuales:

Además de su paradigma de procesamiento, los asistentes virtuales se diferencian por el estilo arquitectónico de software adoptado para su implementación. Esta decisión de diseño de infraestructura impacta directamente en atributos de calidad estandarizados como la mantenibilidad, la escalabilidad y la extensibilidad del sistema [33]. Los principales estilos arquitectónicos utilizados son:

- **Arquitecturas monolíticas:** integran todos los componentes del sistema (procesamiento de lenguaje natural, gestión de diálogo, lógica de negocio y acceso a bases de datos) en una única base de código y unidad de despliegue [34]. Aunque simplifican el desarrollo y la orquestación inicial, presentan un alto nivel de acoplamiento que restringe la escalabilidad horizontal y vuelve técnica y operativamente inviable la integración de modelos heterogéneos en el mismo entorno.
- **Arquitecturas cliente-servidor:** separan físicamente la interfaz de usuario (el canal conversacional) del procesamiento centralizado en el *backend* [35]. Este enfoque clásico mejora la mantenibilidad respecto al monolito y permite actualizar el motor lógico sin alterar la interfaz, pero sigue centralizando la carga computacional, lo cual puede generar cuellos de botella al realizar inferencia con modelos LLM.
- **Arquitecturas orientadas a microservicios y contenedores:** descomponen las funcionalidades del asistente en servicios independientes, altamente cohesivos y débilmente acoplados, que se comunican a través de interfaces de red estándar [36]. En el desarrollo moderno de IA, esta arquitectura se apoya fuertemente en la contenerización [37]. Esto permite que cada servicio (e.g., un motor de inferencia LLM local o un clasificador de intenciones basado en NLU) se empaquete con sus propias dependencias y se despliegue de forma aislada, facilitando la sustitución progresiva de tecnologías sin afectar al resto del sistema.
- **Arquitecturas distribuidas descentralizadas:** extienden el concepto de microservicios para soportar la ejecución de componentes a través de múltiples

nodos lógicos o físicos [35]. Constituyen la base de infraestructura necesaria para desplegar ecosistemas multi-agente complejos, permitiendo que agentes independientes se ejecuten en entornos locales (fuera de la nube) mientras son coordinados por una capa de orquestación centralizada que enruta los mensajes entre los distintos contenedores.

2.1.3 Heterogeneidad y limitaciones actuales

La coexistencia de los paradigmas funcionales y arquitectónicos descritos introduce una heterogeneidad técnica considerable. Mientras que un sistema basado en reglas puede exponer sus servicios a través de *webhooks* simples, un asistente impulsado por un modelo LLM puede requerir el envío continuo de contexto mediante API asíncronas y un servicio analítico suele operar procesando lotes de datos estructurados. Esta divergencia en interfaces, protocolos de comunicación y formatos de mensajería dificulta la integración directa entre componentes y favorece la aparición de silos de información.

En la práctica, muchas plataformas educativas dependen de integraciones *ad-hoc* mediante API propietarias dependientes de un solo proveedor comercial. Esto produce un acoplamiento fuerte, la duplicación de flujos de datos y altos costos de mantenimiento. Tales limitaciones evidencian la necesidad de enfoques arquitectónicos que, mediante capas de orquestación estandarizadas, permitan coordinar múltiples asistentes de forma coherente y modular [24, 5].

2.1.4 Paradigmas de despliegue: computación en la nube vs. computación local

Para comprender el dilema operativo y de privacidad en los asistentes virtuales, es necesario definir los dos entornos principales donde reside el software y se procesan los datos: la nube y el entorno local.

La computación en la nube se refiere al suministro de servicios a través de Internet, donde los recursos (servidores, bases de datos, motores de inferencia de IA) son gestionados por proveedores externos en centros de datos remotos [38]. En el contexto de los asistentes virtuales, la nube permite acceder a modelos de gran escala de forma inmediata, delegando el mantenimiento técnico al proveedor. Sin embargo, genera dependencia tecnológica y obliga a transmitir información institucional hacia servidores de terceros.

Por el contrario, la computación local implica que el software y el procesamiento se ejecuten en infraestructura controlada directamente por la organización o el usuario. Esto otorga una autonomía total sobre la configuración del sistema y elimina la latencia de red externa [39].

La principal diferencia entre ambos paradigmas radica en la soberanía tecnológica y el trayecto que recorren los datos sensibles del estudiante. Como ilustra la Figura 2.2, mientras que el modelo de nube extrae el flujo de datos hacia la infraestructura comercial, el enfoque local permite encapsular la IA dentro de un perímetro seguro y auditable por la institución [15, 40].

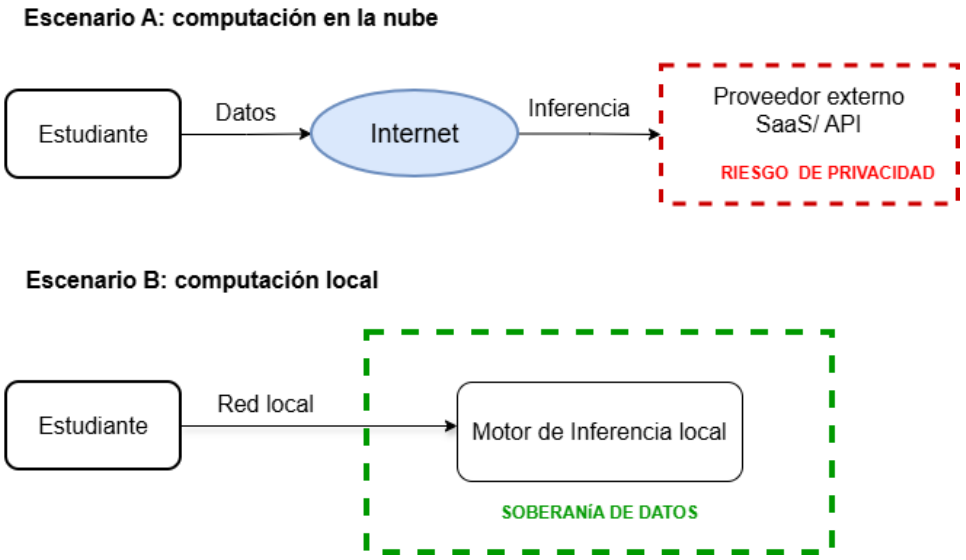


Figura 2.2: Comparativa de flujos de datos y perímetro de seguridad entre paradigmas de computación en la nube y ejecución local.

2.1.5 El dilema de la inferencia: soberanía de datos y privacidad

Si bien el dilema del entorno de despliegue es aplicable a diversos tipos de asistentes virtuales, cobra especial relevancia en los LLM, debido a su alta demanda de cómputo y a la sensibilidad de los datos procesados. Mientras que en los asistentes basados en reglas o NLU clásico el procesamiento es ligero y fácilmente integrable de forma local, los asistentes generativos modernos plantean una decisión crítica respecto a la privacidad del usuario [40].

Los servicios de inferencia en la nube ofrecen acceso a modelos de vanguardia con un esfuerzo de configuración casi nulo. Sin embargo, este enfoque introduce desafíos insalvables respecto a la soberanía de datos, ya que la información del estudiante debe viajar a servidores externos, lo que puede vulnerar normativas de privacidad institucional y regulaciones de protección de datos [15].

En respuesta a esta problemática, la inferencia local —facilitada por motores de ejecución como Ollama [41] y gestionada mediante plataformas de contenerización [42]— permite que los modelos operen de forma agnóstica e independiente. Este paradigma asegura que toda la inteligencia educativa permanezca bajo el control estricto de la institución, permitiendo además que los modelos generativos coexistan con asistentes de arquitecturas más simples bajo una misma infraestructura de red unificada, la cual se mantiene aislada mediante virtualización a nivel de sistema operativo [37].

2.2 Sistemas multi-agente e interoperabilidad

La interoperabilidad se define como la capacidad de múltiples sistemas software, potencialmente heterogéneos, para intercambiar información y utilizarla de manera efectiva sin requerir modificaciones sustanciales en sus implementaciones internas [33]. En el contexto de sistemas inteligentes distribuidos, este concepto implica no solo compatibilidad técnica, sino también alineación semántica, coordinación de protocolos y desacoplamiento arquitectónico.

En entornos educativos modernos, donde coexisten múltiples agentes virtuales especializados —e.g., tutores conversacionales, sistemas de recomendación, motores analíticos o servicios administrativos— la interoperabilidad se convierte en un requisito fundamental para evitar silos funcionales y permitir flujos de trabajo integrados. Sin mecanismos adecuados de integración, cada componente evoluciona de forma aislada, dificultando la reutilización de capacidades y aumentando los costos de mantenimiento.

Desde la perspectiva de la Ingeniería de Software, la interoperabilidad suele abordarse mediante principios de bajo acoplamiento, comunicación asíncrona y separación de responsabilidades, características especialmente relevantes en arquitecturas distribuidas y multi-agente [23].

La interoperabilidad en sistemas multi-agente ha sido abordada tradicionalmente desde dos perspectivas complementarias: (i) la definición de protocolos formales de comunicación entre agentes y lenguajes estandarizados de intercambio de mensajes, y (ii) la coordinación de comportamientos colaborativos mediante mecanismos de orquestación, negociación y cooperación distribuida [23].

Como señala el exhaustivo estudio de Xi et al. [24], la transición de modelos de lenguaje aislados a ecosistemas de agentes autónomos representa un cambio de paradigma fundamental para la resolución de tareas complejas. Recientemente, la irrupción de estos LLM y asistentes cognitivos ha propiciado la aparición de nuevos *frameworks* que facilitan la colaboración entre asistentes conversacionales. No obstante, dichas soluciones tienden a priorizar la coordinación lógica a nivel de aplicación más que la interoperabilidad arquitectónica entre sistemas heterogéneos [43, 44, 45].

2.2.1 *Frameworks* contemporáneos para agentes colaborativos

El auge de los LLM ha impulsado la aparición de *frameworks* diseñados para coordinar múltiples asistentes conversacionales o cognitivos. Estas plataformas buscan facilitar la orquestación de tareas complejas mediante la colaboración de asistentes especializados. Previo a la consolidación de estos *frameworks*, arquitecturas pioneras como CAMEL [46] demostraron la viabilidad de la exploración autónoma mediante la asignación estricta de roles, mientras que AgentVerse [47] evidenció cómo la colaboración multi-agente fomenta comportamientos emergentes superiores a los de un agente individual. No obstante, la mayoría de estas soluciones prioriza la orquestación lógica de tareas a nivel de aplicación más que la interoperabilidad arquitectónica entre sistemas heterogéneos. A continuación se presentan algunos de los *frameworks* más relevantes en la literatura reciente:

- **CrewAI:** *framework* de código abierto orientado a la definición de equipos *(crews)* de agentes basados en LLM que cooperan mediante la asignación explícita de roles, objetivos y tareas encadenadas. Permite modelar flujos secuenciales o jerárquicos donde cada agente asume responsabilidades especializadas (e.g., investigación, redacción o verificación) simplificando la construcción de aplicaciones colaborativas [43]. Sin embargo, su alcance se concentra en la coordinación a nivel de aplicación y no proporciona mecanismos nativos de interoperabilidad a nivel de infraestructura, tales como desacoplamiento tecnológico, mensajería distribuida o integración estandarizada con sistemas externos. En el contexto educativo, esta carencia arquitectónica impide que un ecosistema de agentes tutores interactúe de forma transparente con los repositorios institucionales, provocando que la información generada —como el progreso cognitivo de un estudiante— quede aislada del sistema de control escolar.

- **AutoGen:** propuesto por Microsoft Research, facilita la interacción conversacional entre múltiples agentes y herramientas externas mediante intercambios estructurados de mensajes y control automático del diálogo [44]. Esta aproximación favorece la experimentación y el prototipado rápido de comportamientos colaborativos; no obstante, su arquitectura se orienta principalmente a entornos de investigación y no define estándares formales de integración ni soporte específico para despliegues empresariales o institucionales a gran escala. Para una institución de educación superior, donde la trazabilidad, la seguridad y la auditoría de las interacciones (e.g., en sistemas de asesoría académica automatizada) son requisitos normativos, la dependencia de flujos de mensajes *ad-hoc* dificulta su implementación como un servicio confiable a nivel infraestructura.

- **LangChain Agents y LangGraph:** herramientas que permiten construir flujos de ejecución compuestos por cadenas o grafos de razonamiento que coordinan múltiples agentes, herramientas y servicios externos [48, 49]. Aunque ofrecen gran flexibilidad para el desarrollo de aplicaciones basadas en LLM, su integración se mantiene mayormente a nivel de biblioteca dentro del mismo entorno tecnológico, lo que introduce dependencias fuertes del ecosistema Python y dificulta la interoperabilidad con plataformas desarrolladas en otros lenguajes o arquitecturas distribuidas. Este nivel de acoplamiento representa un obstáculo significativo en las universidades, cuyos sistemas de gestión de aprendizaje y plataformas administrativas suelen estar cimentados sobre arquitecturas empresariales heterogéneas consolidadas —frecuentemente basadas en ecosistemas Java o .NET—, complicando el despliegue de los agentes como servicios modulares y transversales.

- **Plataformas clásicas de agentes (JADE y entornos compatibles con FIPA):**
  En el ámbito tradicional de sistemas multi-agente, plataformas como JADE implementan especificaciones estandarizadas para comunicación y coordinación entre agentes autónomos [50, 23]. Estas soluciones proporcionan fundamentos

sólidos para interacción distribuida y protocolos formales de mensajería; sin embargo, su adopción en ecosistemas modernos basados en LLM es limitada y la integración directa con servicios web contemporáneos o modelos generativos requiere adaptaciones adicionales. Como es operativamente complejo emplear estos entornos para orquestar la nueva generación de asistentes educativos, los cuales demandan una integración fluida y basada en contenedores con motores de inferencia modernos e infraestructuras de IA distribuidas.

- **Semantic Kernel:** SDK desarrollado por Microsoft y diseñado específicamente para integrar LLM con arquitecturas empresariales tradicionales. A diferencia de las soluciones centradas exclusivamente en Python, Semantic Kernel soporta nativamente lenguajes como C# y Java, organizando las capacidades de los agentes a través de *plugins* y funciones nativas [51]. En el ámbito educativo, esta flexibilidad multilingüe facilita la conexión de agentes con los sistemas administrativos heredados de las universidades; sin embargo, al tratarse de un SDK, obliga a que los componentes compartan la misma base tecnológica o de ejecución, lo que impide alcanzar una verdadera interoperabilidad distribuida basada en contenedores y protocolos de red agnósticos.
- **LlamaAgents (LlamaIndex):** es un *framework* que adopta un enfoque de arquitectura orientada a servicios para sistemas multi-agente, donde cada agente opera como un microservicio independiente que se comunica a través de un plano de control centralizado y colas de mensajes [52]. Está fuertemente optimizado para tareas de RAG. Para una institución educativa, esto resulta ideal al momento de conectar asistentes virtuales con bibliotecas digitales o bases de conocimiento institucionales. No obstante, la dependencia de un enrutador central (*control plane*) específico de la plataforma reintroduce un punto único de falla y limita la adopción de protocolos de comunicación abiertos y descentralizados entre agentes heterogéneos.
- **MetaGPT:** es un *framework* multi-agente que modela la colaboración estructurando a los agentes en roles profesionales y obligándolos a seguir Procedimientos Operativos Estándar para coordinar flujos de trabajo complejos [53]. Si bien la estandarización de procesos resulta útil para automatizar flujos administrativos estáticos (como el diseño instruccional de un curso) su rigidez procedimental dificulta la interoperabilidad dinámica requerida en un campus virtual, donde múltiples agentes (tutores, mentores, servicios escolares) deben interactuar y negociar información de manera flexible y no necesariamente predefinida.

El análisis del estado del arte evidencia que, si bien existe un interés creciente por la colaboración entre agentes, las soluciones actuales priorizan la orquestación a nivel de aplicación sobre la interoperabilidad arquitectónica. Esto genera tres limitaciones comunes: dependencia de ecosistemas cerrados (acoplamiento tecnológico), comunicación basada en flujos síncronos y carencia de mecanismos explícitos para integrar servicios no conversacionales (como analítica educativa o sistemas heredados). Estas restricciones resultan críticas en contextos institucionales, donde deben coexistir múltiples plataformas tecnológicas y lenguajes. Para sintetizar estas áreas de oportunidad y justificar el diseño abordado en este trabajo, la Tabla 2.1 contrasta los *frameworks* analizados frente a cinco dimensiones clave. Se hace especial énfasis

en el mecanismo de coordinación, diferenciando los enfoques estáticos actuales del enrutamiento dinámico que plantea esta investigación.

Tabla 2.1: Comparativa de *frameworks* de agentes frente a la arquitectura propuesta

Framework / Plataforma	**Enfoque principal**	**Acoplamiento tecnológico**	**Mecanismo de coordinación y enrutamiento**	**Integración de Sistemas Heredados**
CrewAI	Orquestación de roles	Alto (Python)	Asignación estática de tareas secuenciales o jerárquicas	Limitada (vía herramientas personalizadas)
AutoGen	Diálogo multi-agente	Alto (Python / .NET)	Síncrono, basado en respuestas directas entre agentes	Limitada
LangChain / LangGraph	Flujos lógicos (Grafos)	Alto (ecosistema LangChain)	Grafos de ejecución acoplados a la lógica de la aplicación	Moderada (vía integraciones de biblioteca)
Semantic Kernel	Integración empresarial	Moderado (C#, Java, Python)	Invocación local de funciones a través del núcleo (*kernel*)	Alta (ecosistemas corporativos compartidos)
LlamaAgents	Microservicios	Alto (Python)	Enrutador centralizado dependiente del plano de control propio	Moderada (centrada en recuperación documental)
MetaGPT	Procesos estandarizados	Alto (Python)	Secuencial, guiado por Procedimientos Operativos	Limitada
JADE	Sistema multi-agente puro	Alto (Java)	Mensajería estándar (FIPA-ACL) con registro de páginas amarillas	Difícil (con servicios web modernos)
Arquitectura propuesta	Interoperabilidad arquitectónica	Agnóstico basado en contenedores de Docker	Enrutamiento dinámico por capacidades (orquestador sin estado)	Nativa (diseñada para arquitecturas distribuidas)

Como se observa en la Tabla 2.1, las soluciones actuales tienden a acoplar la lógica de coordinación con el entorno de ejecución (típicamente Python) o a depender de flujos de mensajes predefinidos. La novedad de la arquitectura propuesta radica en su enfoque de interoperabilidad y delegación de responsabilidades: emplea un orquestador central que opera como un enrutador inteligente. Dicho orquestador no centraliza el estado ni el perfil del usuario, sino que recibe la consulta y, basándose en un registro dinámico de capacidades, la deriva al asistente especializado correspondiente. Asimismo, la gestión del conocimiento y la lógica de procesamiento se mantienen aisladas dentro de cada asistente especializado, garantizando una arquitectura verdaderamente distribuida, tolerante a fallos y agnóstica al lenguaje de programación de cada componente individual.

2.3 Orquestación y arquitecturas de memoria en agentes de IA

El desarrollo de sistemas multi-agente basados en LLM ha evolucionado más allá de la simple generación de texto, introduciendo arquitecturas complejas de control y persistencia. Esta sección analiza los mecanismos de gobernanza de flujos de trabajo y las estrategias de gestión de información que permiten la colaboración entre entidades inteligentes.

2.3.1 Taxonomía de la orquestación de agentes

La orquestación se define como el proceso de coordinación, gestión y dirección de las interacciones entre múltiples agentes autónomos para lograr un objetivo común. Dependiendo del nivel de autonomía y la dinámica de decisión, se distinguen tres paradigmas principales:

- **Orquestación estática:** representa el modelo tradicional donde el flujo de ejecución está predefinido mediante grafos acíclicos dirigidos. Los agentes se ejecutan en una secuencia rígida (*chains*), donde la salida de un agente es estrictamente la entrada del siguiente. Aunque ofrece alta predictibilidad, presenta una capacidad limitada de adaptabilidad ante consultas fuera del dominio previsto [48].
- **Orquestación dinámica basada en enrutamiento:** en este paradigma, un nodo central u orquestador actúa como un motor de decisión semántica. Evalúa la intención de la consulta y selecciona, en tiempo de ejecución, al agente experto o herramienta más apta, independientemente de la arquitectura interna del asistente seleccionado [5, 54]. Este enfoque permite una escalabilidad modular, facilitando la integración de nuevos asistentes heterogéneos sin reconfigurar la lógica global.
- **Orquestación autónoma y razonamiento ReAct:** basado en el paradigma *Reasoning and Acting* (ReAct), los agentes ejecutan ciclos iterativos de pensamiento, acción y observación [55]. El agente genera una traza de razonamiento sobre qué paso seguir, ejecuta una acción (como consultar un experto o base de datos), observa el resultado y decide si ha alcanzado la meta pedagógica o si requiere una nueva iteración.

2.3.2 Arquitecturas de memoria en sistemas multi-agente

A diferencia de los modelos de lenguaje aislados que operan sin estado, los ecosistemas de agentes modernos requieren estructuras de memoria para mantener la coherencia semántica y el aprendizaje contextual. La literatura técnica clasifica la memoria de los agentes en tres dimensiones:

1. **Memoria de corto plazo:** se limita a la información inmediata procesable dentro de la ventana de contexto del modelo [5]. En ecosistemas interoperables, el desafío técnico es la transferencia de estado, permitiendo que un agente

herede el contexto de la interacción de otro para garantizar la continuidad pedagógica sin redundancias hacia el estudiante [44].

2. **Memoria de largo plazo:** constituye el mecanismo arquitectónico fundamental para superar las restricciones de retención inherentes a los modelos fundacionales. Se logra mediante el acoplamiento con repositorios externos no volátiles, permitiendo al sistema persistir, indexar y recuperar información histórica de manera asíncrona. Implementaciones destacadas como Voyager [56] han demostrado que acoplar LLM con repositorios externos y bibliotecas de habilidades iterativas permite un aprendizaje continuo, sentando un precedente para la retención de contexto en asistentes educativos. La literatura especializada clasifica estas implementaciones en tres paradigmas predominantes [5]:
   - **Bases de datos estructuradas:** utilizadas para el almacenamiento determinista de estados y metadatos precisos en sistemas distribuidos, como el historial académico o perfiles de usuario institucionales [35].
   - **Grafos de conocimiento:** modelan la información mediante entidades y relaciones explícitas, facilitando la inferencia lógica sobre dependencias conceptuales o estructuras de planes de estudio [27].
   - **RAG:** utiliza motores de búsqueda vectorial para indexar y recuperar conocimiento externo mediante representaciones numéricas de alta dimensionalidad (*embeddings*). Al recibir una consulta, el sistema extrae los fragmentos documentales semánticamente más relevantes y los inyecta dinámicamente en la ventana de contexto del modelo de lenguaje antes de la inferencia [57]. En el ámbito de los asistentes educativos, RAG constituye un mecanismo de memoria a largo plazo indispensable: permite que los agentes fundamenten sus respuestas en repositorios institucionales validados (como planes de estudio, literatura académica o manuales), mitigando el riesgo de alucinaciones y evitando el costoso proceso de reentrenar los modelos locales frente a cada actualización del contenido curricular.
3. **Memoria episódica:** registro histórico de interacciones pasadas. Permite al sistema recordar preferencias del usuario, hitos de aprendizaje o errores previos, facilitando una personalización profunda y adaptativa a lo largo del tiempo [58].

2.3.3 Almacenamiento externo y uso de herramientas

Un componente crítico de la orquestación moderna es la capacidad del agente para interactuar con su entorno mediante el uso de herramientas externas [5]. Esto trasciende la capacidad de procesamiento lingüístico, permitiendo a los agentes consultar bases de datos relacionales, ejecutar código en entornos aislados o persistir estados en sistemas de archivos locales, lo cual es esencial para el seguimiento del progreso académico. La estandarización de estas interacciones es el objetivo de protocolos recientes, que buscan unificar el acceso a datos y herramientas [59].

En el contexto de ecosistemas educativos, la combinación de estos tres pilares —enrutamiento dinámico, memoria vectorial a través de RAG y el uso de herramien-

tas externas— conforma la base teórica necesaria para desacoplar el conocimiento institucional de la lógica conversacional individual de cada asistente, preparando el escenario técnico para modelos de integración estandarizados.

2.4 Protocolos estandarizados para la interoperabilidad de agentes y asistentes

La interoperabilidad en sistemas multi-agente y asistentes inteligentes va más allá del simple intercambio de datos entre servicios heterogéneos: requiere la definición de protocolos que regulen la síntesis de mensajes, la coordinación de acciones y la cooperación entre agentes autónomos para alcanzar objetivos compartidos. Esta interoperabilidad puede entenderse como la capacidad de agentes distribuidos para comunicarse, descubrir capacidades, compartir contexto y coordinar tareas sin depender de implementaciones internas, infraestructura específica o acoplamientos rígidos. El desarrollo histórico de protocolos de comunicación y coordinación en sistemas multi-agente proporciona la base conceptual sobre la cual se han construido las recientes especificaciones orientadas a agentes de IA con LLM.

Los protocolos que se discutirán a continuación fueron seleccionados con base en su relevancia para la interoperabilidad de agentes, su presencia en bibliografía académica o industrial y su aplicabilidad a escenarios multi-agente heterogéneos.

2.4.1 Protocolos clásicos de coordinación y comunicación para sistemas multi-agente

Los primeros avances en interoperabilidad entre agentes autónomos surgieron en el ámbito de los sistemas multi-agente clásicos, donde se definieron protocolos formales para la coordinación de tareas, negociación y comunicación semántica.

El protocolo CNP constituye uno de los mecanismos pioneros de asignación distribuida de tareas. En este modelo, un agente gestor publica una solicitud de ejecución, múltiples agentes presentan propuestas y la tarea se adjudica según criterios de coste o capacidad. Este esquema permite negociación descentralizada, asignación dinámica de recursos y coordinación flexible en entornos distribuidos, siendo ampliamente adoptado como fundamento de arquitecturas cooperativas [60].

Posteriormente, la FIPA estandarizó el lenguaje ACL, que define actos de habla, estructuras de mensajes y semántica formal para el intercambio de información entre agentes. ACL establece primitivas como *request*, *inform* o *propose*, proporcionando un marco común para garantizar interoperabilidad sintáctica y semántica entre implementaciones heterogéneas [61].

Sobre estas especificaciones surgieron plataformas de desarrollo compatibles con FIPA, entre las que destaca JADE, el cual implementa servicios de mensajería, descubrimiento de agentes y gestión del ciclo de vida conforme a los estándares ACL. JADE facilita la construcción de sistemas multi-agente interoperables y ha sido utilizada extensamente tanto en investigación como en aplicaciones industriales [50].

Estos protocolos y plataformas sentaron las bases conceptuales de la interoperabilidad entre agentes autónomos, introduciendo principios de comunicación estandarizada, coordinación distribuida y desacoplamiento funcional que continúan influyendo en los protocolos modernos orientados a asistentes inteligentes [23].

2.4.2 Protocolos emergentes para agentes de IA y asistentes colaborativos

La necesidad de interoperabilidad entre asistentes virtuales y agentes de IA contemporáneos ha impulsado el desarrollo de protocolos que trascienden el simple intercambio sintáctico de mensajes e incorporan mecanismos de descubrimiento dinámico de capacidades, gestión distribuida de tareas, compartición de contexto y coordinación entre plataformas heterogéneas. A diferencia de los enfoques clásicos centrados exclusivamente en lenguajes formales de comunicación, estas propuestas integran consideraciones prácticas de despliegue web, escalabilidad y compatibilidad con infraestructuras modernas basadas en servicios.

OFP constituye un estándar abierto impulsado por la *Open Voice Interoperability Initiative* dentro de la *Linux Foundation AI & Data Foundation*, orientado a la coordinación de múltiples agentes y usuarios dentro de un mismo espacio conversacional compartido [62]. OFP define el concepto de "cedo la palabra" (*yield the floor*), en el que distintos participantes pueden intercambiar turnos, delegar acciones y colaborar sobre tareas comunes mediante eventos estructurados en formato JSON, independientes del transporte subyacente (HTTP, WebSockets u otros). Este diseño favorece la neutralidad tecnológica, la concurrencia entre asistentes y la integración de asistentes desarrollados con diferentes *frameworks*. Además, la estandarización de la interoperabilidad de asistentes conversacionales ha sido abordada de forma complementaria por iniciativas formales como el estándar IEEE P2863 [63], lo que refuerza la relevancia de OFP como base para ecosistemas multi-agente.

El protocolo A2A propone un mecanismo de comunicación horizontal entre agentes autónomos, permitiendo el descubrimiento de capacidades mediante descriptores estructurados y la coordinación de tareas colaborativas a través de interfaces interoperables basadas en JSON-RPC sobre HTTPS [64]. Este protocolo prioriza la cooperación entre agentes desarrollados por distintos proveedores o tecnologías, facilitando escenarios donde múltiples agentes deben delegar subtareas o intercambiar resultados de manera dinámica. En contraste con soluciones acopladas a bibliotecas específicas, A2A se concibe como una especificación independiente del lenguaje de implementación.

De forma complementaria, el protocolo MCP estandariza la interacción de los agentes con herramientas externas, fuentes de datos y servicios auxiliares [59]. Mientras A2A aborda la comunicación entre agentes pares, MCP se enfoca en la interoperabilidad vertical agente–herramienta, definiendo interfaces uniformes para consultar API, ejecutar funciones o recuperar información contextual. Esta separación de responsabilidades permite combinar ambos protocolos dentro de arquitecturas modulares.

El protocolo ACP adopta una aproximación ligera basada en principios RESTful para el intercambio de mensajes asincrónicos entre agentes distribuidos [65]. Al apo-

yarse en convenciones HTTP ampliamente adoptadas, ACP simplifica la integración con infraestructuras web existentes y favorece la escalabilidad horizontal, reduciendo la dependencia de entornos o bibliotecas específicas.

Finalmente, el protocolo ANP extiende la interoperabilidad hacia escenarios de redes amplias de agentes, incorporando mecanismos de descubrimiento, identidad descentralizada y comunicación segura entre participantes que no se conocen previamente [66]. Este enfoque resulta pertinente para ecosistemas abiertos o federados, donde los agentes pueden incorporarse o retirarse dinámicamente del sistema.

En conjunto, estas propuestas evidencian una transición desde modelos cerrados o dependientes de *frameworks* hacia especificaciones abiertas y orientadas a servicios. La literatura reciente y la Tabla 2.2 muestran que no existe un único estándar que resuelva todas las dimensiones de la interoperabilidad. Por el contrario, los protocolos tienden a especializarse: MCP habilita la comunicación vertical (agente-herramienta), A2A facilita la cooperación horizontal (agente-agente) y OFP se enfoca en la orquestación de turnos conversacionales [59, 64, 62]. En consecuencia, el diseño de ecosistemas multi-asistente modernos requiere la coexistencia de estos estándares para lograr una interoperabilidad arquitectónica completa.

Tabla 2.2: Comparativa de protocolos para la interoperabilidad en sistemas multi-agente

Protocolo / Estándar	**Enfoque principal**	**Paradigma de comunicación**	**Tipo de interacción**	**Formato de mensajería**
FIPA-ACL	Estandarización semántica clásica (actos de habla)	Mensajería asíncrona estructurada	Agente ↔ Agente	FIPA-SL / XML
CNP	Asignación distribuida de tareas por subasta	Negociación y propuestas	Gestor ↔ Agente	Dependiente de implementación
OFP	Coordinación conversacional y cesión de turnos	Basado en eventos (neutral al transporte)	Multi-agente en espacio compartido	JSON
A2A	Descubrimiento de capacidades y cooperación horizontal	Peticiones estructuradas (RPC)	Agente ↔ Agente	JSON-RPC
MCP	Integración con servicios externos y fuentes de datos	Interfaz de consulta y ejecución	Agente ↔ Herramienta	JSON
ACP	Intercambio asincrónico ligero y escalable	RESTful (basado en recursos)	Agente ↔ Agente	JSON / HTTP
ANP	Descubrimiento e identidad descentralizada en redes abiertas	Punto a punto (P2P) con canales cifrados	Agente ↔ Agente (red abierta)	JSON-LD / DID

La literatura reciente muestra que no existe un único protocolo que resuelva todas las dimensiones de interoperabilidad para asistentes; más bien, los protocolos tienden a especializarse según el dominio de interacción. Por ejemplo, mientras que MCP se centra en la comunicación agente-herramienta para el acceso a datos [59], A2A habilita la comunicación orientada a tareas entre agentes en el mismo nivel de

colaboración [64] y OFP se orienta a la coordinación del turno conversacional en contextos multi-agente [62]. Por su parte, ACP y ANP proporcionan mecanismos complementarios de mensajería y descubrimiento descentralizado en redes a gran escala [65, 66]. Estos protocolos no son excluyentes, sino que operan como una pila de capas que, al coexistir, permiten construir ecosistemas de agentes verdaderamente interoperables.

2.4.3 Protocolos de transporte y mensajería

Para que los protocolos operen de forma fiable, requieren mecanismos de transporte que aseguren la entrega confiable y el desacoplamiento de componentes en entornos distribuidos. Entre las estrategias de comunicación técnica, se destacan:

- **gRPC:** implementación del paradigma de Llamada a Procedimiento Remoto (RPC) desarrollada por Google, basada en *Protocol Buffers* [67], un formato de serialización binaria de alto rendimiento que permite una transferencia de datos eficiente entre microservicios. Es un estándar ideal para comunicaciones internas donde la latencia es crítica. Sin embargo, su implementación exige un acoplamiento estricto mediante la definición previa de esquemas (*contracts*) compartidos entre los agentes que interactúan.
- HTTP/REST y WebSockets: constituyen los estándares de comunicación predominantes en la Web. REST favorece la interoperabilidad mediante interfaces uniformes basadas en recursos accesibles vía URI, siguiendo el modelo arquitectónico de Fielding [68]. Por su parte, WebSockets proporcionan canales de comunicación persistentes y bidireccionales, indispensables para intercambios conversacionales de baja latencia entre el asistente y el usuario en tiempo real.
- Servicios de mensajería asíncrona: basados en colas de mensajes y el patrón de arquitectura *publish/subscribe* (implementaciones como AMQP, MQTT o Apache Kafka), estos servicios permiten el desacoplamiento temporal y lógico entre productores y consumidores de mensajes. Este enfoque es un pilar fundamental para la escalabilidad y elasticidad de sistemas multi-agente, permitiendo que la orquestación soporte picos de carga sin comprometer la integridad del flujo pedagógico o la disponibilidad de los servicios [69, 36].

Aunque estos mecanismos no definen la semántica interna del mensaje, proporcionan la infraestructura de red sobre la cual los protocolos de alto nivel pueden intercambiar información de manera eficiente y escalable.

2.4.4 Contenerización como habilitador de interoperabilidad

La interoperabilidad técnica en sistemas multi-agente no depende únicamente de los protocolos de comunicación, sino también de la capacidad de desplegar componentes heterogéneos sin conflictos de entorno. En el desarrollo de software moderno, la contenerización ha surgido como un cambio de paradigma frente a la virtualización tradicional. Mientras que las máquinas virtuales encapsulan un sistema operativo

completo, los contenedores comparten el núcleo del sistema anfitrión, permitiendo ejecutar procesos de forma aislada con una sobrecarga de recursos mínima [37].

En este contexto, Docker se posiciona como el estándar de la industria para la implementación de esta tecnología. Docker posibilita empaquetar un asistente virtual —junto con todas sus bibliotecas, tiempos de ejecución y configuraciones específicas— dentro de una unidad estandarizada denominada contenedor. Este empaquetado garantiza la portabilidad, permitiendo que el asistente funcione de manera idéntica tanto en el entorno de desarrollo como en los servidores de producción, independientemente de la configuración subyacente del sistema operativo [70].

Desde la perspectiva de la interoperabilidad, Docker actúa como un habilitador infraestructural al facilitar arquitecturas basadas en microservicios. Al encapsular a cada asistente como una unidad de despliegue desacoplada, se elimina el acoplamiento tecnológico: un asistente tutor desarrollado en Java puede coexistir con un motor de inferencia LLM escrito en Python dentro de la misma infraestructura, comunicándose únicamente mediante protocolos estandarizados de red. Esta separación de responsabilidades entre la infraestructura de ejecución (aislamiento) y el mecanismo de comunicación (protocolos) es el pilar fundamental que permite construir ecosistemas educativos altamente escalables, mantenibles y tecnológicamente diversos [36, 34].

2.5 Estado actual de la integración de asistentes educativos

La adopción de asistentes virtuales en plataformas de gestión del aprendizaje ha crecido de forma sostenida. Sin embargo, la mayoría de estas soluciones operan como componentes aislados, lo que limita la verdadera interoperabilidad entre múltiples asistentes heterogéneos. A continuación, se revisan las limitaciones de las integraciones tradicionales frente a los enfoques modernos necesarios para orquestar asistentes de IA.

2.5.1 Limitaciones de integración en plataformas educativas actuales

Actualmente, plataformas modernas como Moodle o edX —i.e., los sistemas de gestión de aprendizaje— permiten incorporar servicios externos mediante estándares bien establecidos [71, 72]. Estos mecanismos están diseñados para conectar herramientas estáticas a un sistema central, compartiendo datos básicos como la identidad del estudiante o sus calificaciones [73]. El problema fundamental radica en que estos estándares posibilitan una comunicación punto a punto, pero no ofrecen mecanismos nativos para orquestar asistentes virtuales independientes. Cuando una institución integra un chatbot mediante estos métodos convencionales, el asistente queda encapsulado en su propio microservicio.

Como resultado, cuando coexisten varios asistentes especializados, estos operan como silos funcionales: los datos, estados conversacionales y modelos de cada asistente

permanecen aislados. No cuentan con la capacidad de compartir contexto conversacional, delegar tareas o interactuar entre sí de forma autónoma. Esta limitación técnica en el estado del arte obstaculiza la creación de sistemas colaborativos y subraya la necesidad de un modelo arquitectónico superior, capaz de enrutar interacciones entre IA heterogéneas sin depender de los conductos rígidos de un sistema de gestión de aprendizaje.

2.5.2 Implementaciones actuales: agentes comerciales en la nube vs. plataformas de inferencia local

En la intersección entre la IA y la Interacción Humano-Computadora, los agentes generativos se definen como entidades computacionales impulsadas por LLM con capacidades de razonamiento y planificación [5]. A diferencia de los *chatbots* tradicionales basados en flujos rígidos, estos agentes adaptan su comportamiento dinámicamente y utilizan herramientas externas manteniendo el contexto temporal mediante esquemas de memoria [58]. El despliegue de estos agentes se ha bifurcado en dos paradigmas: ecosistemas cerrados en la nube y plataformas de inferencia local.

Agentes generativos comerciales y arquitecturas cerradas

La adopción de agentes generativos ha estado dominada por corporaciones tecnológicas que ofrecen soluciones centralizadas mediante API. Aunque representan la vanguardia en capacidades cognitivas, operan bajo modelos cerrados:

- **ChatGPT (OpenAI):** sustentado en la arquitectura de la familia GPT-4, este entorno estandarizó el paradigma de agentes configurables, permitiendo la especialización de tareas mediante instrucciones de sistema en lenguaje natural y la recuperación de conocimiento estructurado. Su relevancia técnica reside en la integración nativa de un *sandbox* para la ejecución de código y análisis de datos, lo que permite al agente realizar cómputo determinista y visualización sobre resultados de inferencia probabilística [74].
- **Gemini (Google):** diseñado bajo una arquitectura nativamente multimodal, este modelo no depende de codificadores externos para procesar modalidades distintas. A diferencia de los sistemas que concatenan modelos de visión y lenguaje, la arquitectura de Gemini permite realizar razonamiento intermodal sobre el mismo espacio latente. Para un agente educativo, esto facilita la interpretación de recursos pedagógicos complejos (como diagramas, video-clases o audio) sin la latencia ni la pérdida de información que implica la fragmentación en módulos de procesamiento separados [75].
- **Claude (Anthropic):** esta plataforma se distingue por su arquitectura optimizada para la coherencia en contextos masivos y la alineación mediante **constitutional AI**, un método que permite restringir el comportamiento del modelo basándose en un conjunto explícito de principios éticos. Su capacidad técnica más relevante para agentes educativos es la gestión de ventanas de contexto extendidas (hasta 10^6 tokens), lo que permite al agente realizar un análisis longitudinal de grandes corpus documentales (como libros de texto

completos o historiales bibliográficos) manteniendo la fidelidad a la fuente y mitigando la pérdida de información por truncamiento [76].

Si bien estas plataformas facilitan el despliegue rápido, imponen una dependencia tecnológica directa. Para las instituciones educativas, este paradigma exige que la información académica sea transmitida a servidores externos, lo cual contraviene normativas de soberanía de datos y privacidad institucional [15, 40].

Ecosistemas de inferencia local, cuantización y especialización

Como contrapropuesta a la centralización en la nube, han surgido plataformas orientadas a la inferencia local que democratizan el uso de modelos de pesos abiertos. La ventaja fundamental de este enfoque es la privacidad: el procesamiento de los datos se encapsula estrictamente dentro de la infraestructura de la institución, para evitar la dependencia de servicios comerciales en la nube y garantizar la soberanía de los datos. Para comprender cómo es viable ejecutar estos modelos localmente y por qué es necesario orquestarlos, es fundamental analizar su ciclo de vida y los mecanismos de optimización que emplean.

Paradigmas de entrenamiento y especialización: el desarrollo de un modelo de lenguaje comienza con una fase de pre-entrenamiento, donde se obtiene un modelo base capaz de predecir secuencias de texto, aunque inicialmente no dispone de las habilidades necesarias para una interacción fluida. Para transformarlos en asistentes virtuales útiles, los modelos se someten a procesos de alineación que los capacitan para seguir instrucciones o mantener diálogos naturales. Estos procesos incluyen técnicas de ajuste fino supervisado y métodos de aprendizaje mediante retroalimentación humana [77], así como estrategias más recientes basadas en la optimización directa de preferencias del usuario [78]. Esta fase de especialización es la que permite obtener modelos capaces de ejecutar tareas precisas, ya sea mediante la orientación a seguir instrucciones o mediante la optimización para el diálogo. Como resultado de esta fase de especialización, los modelos LLM se dividen en dos categorías principales:

- **Modelos generalistas**: entrenados sobre *corpus* amplios para mantener interacciones fluidas y razonamiento lógico general. Ejemplos de código abierto como la familia LLaMA 3 establecen el estándar para tareas de tutoría y comprensión de lectura [31].
- **Modelos de dominio específico**: optimizados sobre conjuntos de datos altamente técnicos. Modelos como la familia *Qwen* orientados a código superan métricas de referencia en generación y depuración algorítmica [79]. En un entorno educativo, esta especialización técnica evidencia que un solo modelo no puede resolver todas las tareas, requiriendo un enfoque multi-agente.

Cuantización y reducción de la huella computacional: el principal desafío para el despliegue de asistentes generativos fuera de la nube es el elevado consumo de memoria VRAM. Los modelos operan nativamente con precisión de punto flotante de 16 bits, lo cual demanda hardware de nivel empresarial. Para solventar este requerimiento, los ecosistemas de inferencia local emplean técnicas de cuantización, que reducen la precisión numérica de los pesos neuronales (a 8, 4 o incluso 2 bits).

Este proceso utiliza formatos estandarizados como el formato unificado generado por GPT [80] o la cuantización de pesos basada en la activación (AWQ) [81]. Esta técnica disminuye drásticamente los requerimientos computacionales, reduciendo el consumo de memoria hasta en un 70 % con una degradación mínima en la precisión cognitiva [82, 81]. Es precisamente la cuantización lo que permite la coexistencia de múltiples agentes especializados en los servidores institucionales.

Ollama como habilitador agnóstico de inferencia: en este paradigma, herramientas como Ollama actúan como componentes infraestructurales críticos. Este ecosistema se fundamenta en *llama.cpp*, un motor de inferencia altamente optimizado para procesar modelos de lenguaje en hardware de consumo mediante técnicas de cómputo eficiente [83]. Ollama abstrae la complejidad de la gestión de hardware, asignando dinámicamente recursos entre la CPU y la GPU, permitiendo la ejecución de modelos cuantizados de forma fluida.

Para fines de interoperabilidad, el mayor aporte de Ollama es la encapsulación de la ejecución del modelo local bajo una API RESTful estandarizada [41]. Esta capa de abstracción permite que cualquier contenedor externo realice consultas sin acoplarse a la arquitectura interna del modelo. Este desacoplamiento garantiza que los asistentes virtuales sean agnósticos al motor de lenguaje subyacente, facilitando la sustitución o actualización de modelos sin requerir modificaciones en el código de la plataforma educativa. A pesar de que herramientas como Ollama resuelven eficientemente el despliegue privado de modelos individuales como Qwen o LLaMA, presentan una carencia arquitectónica en el contexto de sistemas complejos: no poseen una capa nativa de orquestación multi-agente. Es decir, facilitan la ejecución simultáneamente tres modelos especializados diferentes, pero no proveen los protocolos de red ni las interfaces necesarias para que dichos modelos interactúen, se deleguen tareas o compartan contexto entre sí de forma autónoma. Esta limitación subraya la necesidad de un *framework* de interoperabilidad superior que actúe como enrutador inteligente.

2.5.3 Brecha de investigación

La revisión del estado del arte evidencia que la interoperabilidad en entornos educativos ha sido abordada de manera fragmentada. Por un lado, los estándares tradicionales resuelven la integración de herramientas y el registro de datos [71, 73], pero no están diseñados para la orquestación de asistentes conversacionales. Por otro lado, las plataformas comerciales de IA generativa ofrecen altas capacidades de razonamiento, pero vulneran la soberanía de los datos al operar en ecosistemas cerrados [15, 40].

Si bien los motores de inferencia local solucionan la privacidad y el acoplamiento a servicios en la nube [41], presentan una limitación arquitectónica fundamental: operan como componentes de procesamiento aislados, sin mecanismos estandarizados para la delegación de tareas y la transferencia de contexto conversacional entre múltiples asistentes.

En consecuencia, se identifica una brecha arquitectónica crítica: la necesidad de un modelo de interoperabilidad agnóstico que permita orquestar asistentes heterogéneos

sin depender de servicios propietarios en la nube. Para resolver lo antes planteado se diseñó una solución basada en contenedores [42] que implementa protocolos de comunicación abiertos, como el OFP [62], permitiendo a las instituciones educativas enrutar dinámicamente consultas e integrar IA preservando la escalabilidad, el bajo acoplamiento y la total soberanía de sus datos.

Capítulo 3

Análisis y diseño del modelo

En este capítulo se presenta el análisis de requerimientos y el diseño detallado del modelo de interoperabilidad propuesto. Se parte de los objetivos y del estado del arte analizado en el Capítulo 2 —donde se identificó la falta de estándares de comunicación y la fragmentación técnica entre asistentes— para definir los requerimientos funcionales y no funcionales, la arquitectura general del sistema y los diagramas de diseño que guiarán la implementación posterior.

El presente capítulo se organiza de la siguiente manera: primero, se establecen los requerimientos funcionales y no funcionales que rigen el comportamiento del sistema (cf. Sección 3.1); posteriormente, se describe la especificación del modelo mediante diagramas de casos de uso para identificar a los actores y sus interacciones (cf. Sección 3.2). Seguido de esto, se presenta la arquitectura basada en microservicios (cf. Sección 3.3) y el diseño detallado del protocolo de comunicación OFP mediante diagramas de secuencia (cf. Sección 3.4). Finalmente, se detallan las herramientas tecnológicas y el entorno de desarrollo seleccionados para la materialización del modelo (cf. Sección 3.5).

3.1 Requerimientos del modelo

Antes de proceder con el diseño arquitectónico, es fundamental establecer las capacidades operativas y las restricciones técnicas que definirán el éxito del modelo. Esta fase de análisis permite traducir las necesidades de interoperabilidad en especificaciones formales, identificando las funciones críticas necesarias para resolver la fragmentación en los ecosistemas de asistentes virtuales educativos. En esta sección se establecen los requerimientos funcionales y no funcionales, categorizados según su prioridad, para asegurar un desarrollo con rigor técnico que sirva de base para un diseño modular y escalable.

3.1.1 Requerimientos funcionales

Los requerimientos funcionales definen las capacidades operativas esenciales que el modelo debe poseer para resolver la heterogeneidad y falta de coordinación en los asistentes educativos. Dado que la solución propuesta se fundamenta en un paradigma

de orquestación inteligente, resulta imperativo especificar cómo el sistema gestionará el flujo de información, desde la interpretación de la consulta hasta la respuesta final. En este sentido, la clasificación automática de intenciones y el mecanismo de delegación dinámica hacia asistentes especializados se establecen como las funciones críticas que permiten la interoperabilidad del ecosistema, las cuales se detallan en la Tabla 3.1 (los requerimientos funcionales están denotados por RF).

Tabla 3.1: Especificación de requerimientos funcionales del modelo.

ID	**Requerimiento**	**Especificación**
RF01	Clasificación de intenciones	El sistema debe analizar la consulta del usuario mediante un modelo de lenguaje (LLM) para identificar su intención, con base en el registro de capacidades disponibles, y seleccionar el asistente virtual o los asistentes virtuales aptos para generar una respuesta.
RF02	Delegación dinámica	El sistema debe enviar la consulta del usuario al adaptador del asistente virtual correspondiente, con base en los metadatos declarados en su manifiesto, para asegurar que la solicitud sea procesada por el componente adecuado.
RF03	Gestión de contexto	El sistema debe transferir el contexto inmediato de la conversación actual entre el orquestador y los asistentes virtuales involucrados, cada vez que se delegue una consulta, para mantener la coherencia semántica de la respuesta generada.
RF04	Traducción de protocolos	El sistema debe convertir los mensajes del formato nativo de cada asistente virtual al estándar de mensajería de OFP, antes de enrutarlos a través del orquestador, para garantizar la interoperabilidad sintáctica entre componentes heterogéneos.
RF05	Paralelismo de ejecución	El sistema debe consultar de forma simultánea a múltiples asistentes virtuales, cuando la estrategia de orquestación dinámica determine que la consulta requiere más de una fuente de conocimiento, para reducir el tiempo total de respuesta.

Tabla 3.1: Especificación de requerimientos funcionales del modelo (continuación).

ID	Requerimiento	Especificación
RF06	Descubrimiento y registro	El sistema debe exponer terminales (*endpoints*) que permitan registrar nuevos asistentes virtuales y leer sus manifiestos de capacidades en tiempo de ejecución, sin requerir el reinicio del orquestador, para habilitar la incorporación dinámica de nuevos componentes al ecosistema.

3.1.2 Requerimientos no funcionales

Los requerimientos no funcionales establecen las restricciones y atributos de calidad que garantizan la viabilidad técnica y operativa del modelo propuesto. A diferencia de las capacidades funcionales, estos requerimientos se centran en la estabilidad operativa de la infraestructura y la eficiencia de la comunicación entre componentes. En el presente diseño, se prioriza el desacoplamiento arquitectónico y la portabilidad mediante el uso de tecnologías de contenerización, asegurando que el sistema sea capaz de operar en entornos heterogéneos con una latencia mínima y un alto grado de aislamiento, tal como se especifica en la Tabla 3.2 (los requerimientos no funcionales están denotados por RNF).

Tabla 3.2: Especificación de requerimientos no funcionales del modelo.

ID	Atributo	Especificación
RNF01	Portabilidad	El sistema debe ser desplegable en cualquier infraestructura compatible con contenedores Docker, garantizando la paridad entre los entornos de desarrollo y producción.
RNF02	Bajo acoplamiento	La arquitectura debe basarse en una comunicación agnóstica al lenguaje de programación, permitiendo integrar o remover asistentes sin modificar el núcleo del orquestador.
RNF03	Latencia	El proceso de clasificación y delegación debe optimizarse para no exceder un retardo de 500ms adicionales al tiempo de respuesta intrínseco de los modelos de lenguaje.
RNF04	Escalabilidad	El modelo debe permitir la orquestación de múltiples asistentes concurrentes mediante una red interna de microservicios, sin degradar el rendimiento del sistema.

Tabla 3.2: Especificación de requerimientos no funcionales del modelo (continuación).

ID	Atributo	Especificación
RNF05	Aislamiento y seguridad perimetral	Cada asistente virtual debe ejecutarse en un entorno virtualizado independiente (contenedor) para prevenir conflictos de dependencias y limitar la superficie de ataque entre componentes, constituyendo la línea base de seguridad de la infraestructura.
RNF06	Privacidad y soberanía	El procesamiento semántico debe realizarse estrictamente en la infraestructura local, impidiendo la fuga de datos educativos a servicios de terceros en la nube.
RNF07	Interoperabilidad	El sistema debe garantizar que los asistentes virtuales heterogéneos intercambien información mediante el estándar de mensajería OFP y que dicha información sea correctamente interpretada y utilizada por el asistente receptor, preservando la semántica de la consulta original independientemente de su tecnología, lenguaje de implementación o formato de datos nativo.

Es importante precisar que la seguridad integral del sistema —entendida como autenticación de usuarios, autorización granular, cifrado de datos en tránsito y en reposo, y auditoría formal ante amenazas— se declara fuera del alcance de la presente investigación, se aborda únicamente el uso de contenedores para el aislamiento de procesos (RNF05) y de la ejecución local de los modelos de lenguaje (RNF06), lo cual mitiga riesgos elementales de fuga de datos y contaminación cruzada entre asistentes. La definición de los requerimientos funcionales y no funcionales constituye el marco de referencia técnico que garantiza la viabilidad del sistema. Una vez establecidas estas capacidades mínimas y restricciones de calidad, se cuenta con la base necesaria para proyectar la solución arquitectónica. En el siguiente apartado, se detalla el diseño del modelo de interoperabilidad, donde estos requerimientos se traducen en un diseño conceptual y funcional, definiendo los actores involucrados y las reglas de interacción que permiten la coexistencia de múltiples asistentes virtuales en un ecosistema unificado.

3.2 Diseño del modelo de interoperabilidad

El modelo propuesto se concibe como un ecosistema de código abierto, cuyo propósito fundamental es resolver la fragmentación técnica entre asistentes virtuales educativos. A diferencia de otras soluciones, este modelo permite la integración de

asistentes virtuales externos —frecuentemente provenientes de repositorios de código abierto o desarrollos independientes— y se coordinan bajo un marco de comunicación unificado.

El modelo actúa como una capa de abstracción que gestiona el ciclo de vida de la interacción educativa, permitiendo que el estudiante acceda a múltiples dominios de conocimiento (Matemáticas, Física, etc.) de forma transparente. En esta sección se definen las interacciones primordiales del sistema y los límites funcionales que rigen el intercambio de información entre los actores y los asistentes federados.

3.2.1 Diagrama de Casos de Uso

Una vez identificado el alcance del modelo, la Figura 3.1 ilustra las interacciones entre los actores y las funciones principales del sistema. A continuación, se detallan los elementos que componen este modelo funcional.

Identificación de actores

Se han definido tres actores principales que interactúan con el ecosistema de interoperabilidad:

- **Estudiante (actor principal):** representa al usuario final que demanda servicios educativos. Su interacción es transparente respecto a la complejidad de la red de asistentes.
- **Administrador (actor secundario):** responsable de la gestión técnica del modelo, incluyendo la federación de nuevos asistentes virtuales y la configuración de sus capacidades pedagógicas.
- **Asistente Externo (actor de sistema):** representa a los asistentes virtuales heterogéneos (e.g., asistentes de GitHub o API de LLM) que proveen el conocimiento especializado y actúan como servidores de tareas delegadas.

Análisis de los Casos de Uso principales

El comportamiento del sistema de interoperabilidad se articula a través de los siete casos de uso principales descritos en la Tabla 3.3, los cuales se relacionan directamente con los requerimientos funcionales y no funcionales especificados en las Secciones 3.1 y 3.2, garantizando la trazabilidad entre el análisis, el diseño y la posterior validación del sistema.

Tabla 3.3: Especificación de los Casos de Uso del modelo de interoperabilidad.

ID	Caso de uso	Descripción	Requerimiento relacionado
CU-01	Realizar consulta educativa	Caso de uso principal que representa la entrada lingüística del Estudiante al sistema y que detona obligatoriamente (≪include≫) la clasificación de la intención.	RF01
CU-02	Clasificar intención	El orquestador analiza semánticamente la consulta para identificar el dominio de conocimiento y determinar la ruta óptima.	RF01
CU-03	Delegar tarea	El orquestador transfiere la carga de trabajo, junto con el contexto conversacional, al Asistente Externo más apto mediante protocolos de mensajería estandarizados.	RF02, RF04
CU-04	Orquestar respuesta colaborativa	El orquestador consolida, filtra y formatea la información devuelta por uno o varios Asistentes Externos, para construir una respuesta unificada que se devuelve al flujo principal de la consulta.	RF03, RF05
CU-05	Registrar nuevo asistente	El Administrador da de alta un nuevo Asistente Externo en el ecosistema, configurando sus parámetros de red y credenciales de acceso.	RF06
CU-06	Configurar manifiesto	El Administrador define los metadatos de interoperabilidad (palabras clave, rol pedagógico, capacidades del modelo) que permiten al orquestador descubrir y seleccionar al asistente adecuado.	RF06

Tabla 3.3: Especificación de los Casos de Uso del modelo de interoperabilidad (continuación).

ID	Caso de uso	Descripción	Requerimiento relacionado
CU-07	Monitorizar el sistema	El Administrador supervisa la latencia, el estado de los contenedores y el flujo de mensajes entre la capa de orquestación y los nodos periféricos.	RNF03, RNF05

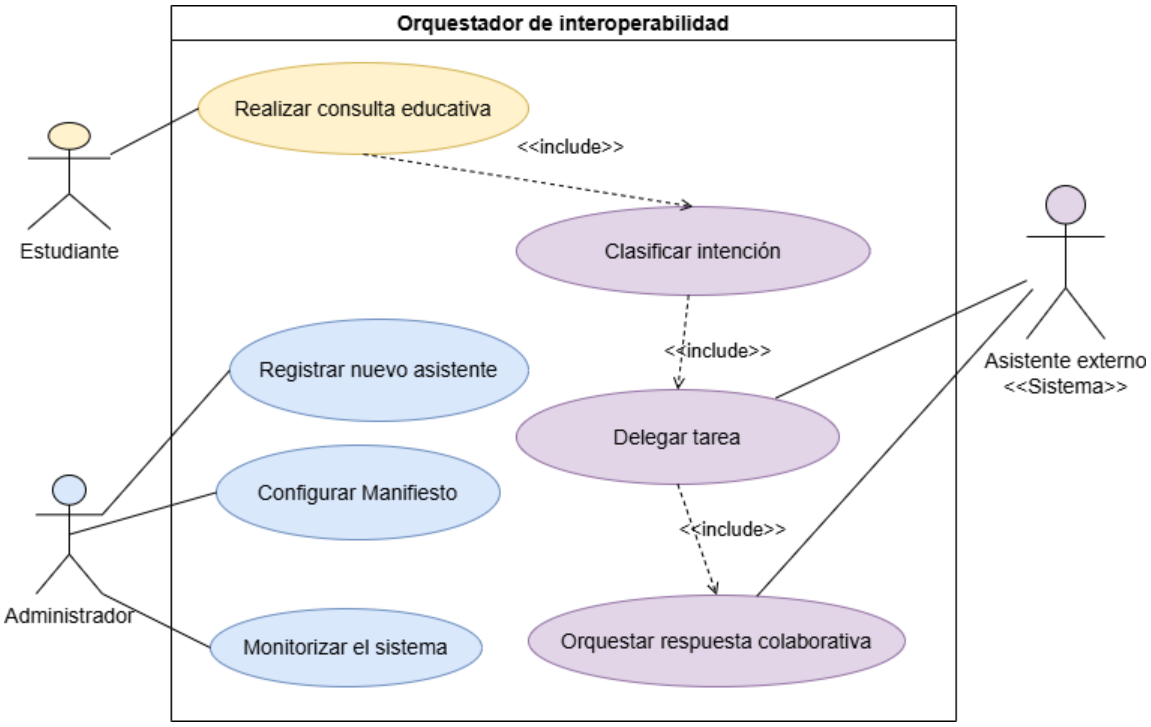


Figura 3.1: Diagrama de Casos de Uso del modelo de interoperabilidad.

3.3 Arquitectura de software del modelo

La arquitectura de software se define como el conjunto de estructuras necesarias para razonar sobre un sistema, el cual comprende los elementos de software, las propiedades externamente visibles de esos elementos y las relaciones entre ellos [84]. La arquitectura del modelo propuesto se fundamenta en un diseño orientado en capas, lo cual permite gestionar la heterogeneidad de los asistentes virtuales mediante el desacoplamiento de sus componentes. Como se ilustra en la Figura 3.2, el modelo separa la capa de consumo de la capa de interoperabilidad, garantizando que el orquestador actúe como un puente estandarizado entre el usuario y los asistentes especializados.

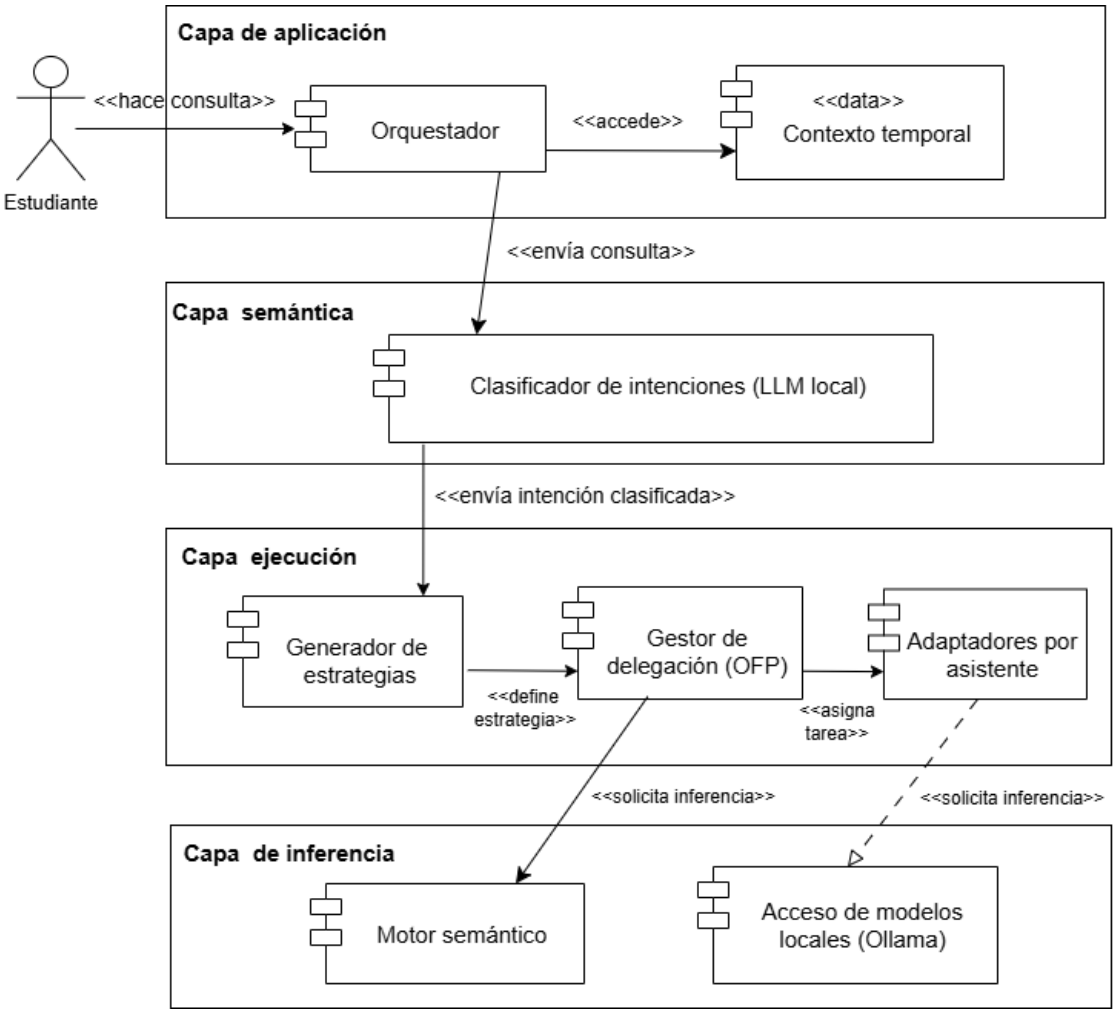


Figura 3.2: Arquitectura lógica del modelo de interoperabilidad basada en capas.

Como se observa en la Figura 3.2, el modelo de interoperabilidad se organiza mediante una estructura jerárquica que garantiza la especialización de tareas y la escalabilidad del sistema:

1. **Capa de aplicación:** constituye el punto de interacción frontal con el Estudiante (actor principal). En este nivel, el orquestador funciona como una interfaz de entrada sin estado, gestionando el contexto temporal mediante un componente de datos activo (*«data»*) que preserva la trazabilidad de la consulta educativa en curso.
2. **Capa semántica:** representa el estrato de interpretación cognitiva. Aquí, el clasificador de intenciones procesa la entrada del usuario para dotarla de significado, determinando la naturaleza de la solicitud antes de ser delegada a la capa de ejecución.
3. **Capa de ejecución:** es el núcleo operativo del modelo. En este nivel se integran los componentes críticos para la orquestación técnica:
4. - **Generador de estrategias:** define el flujo de trabajo (secuencial o paralelo) basándose en la intención clasificada.
   - **Gestor de delegación (OFP):** coordina el envío de tareas bajo el contra-

to estandarizado del protocolo OFP [62], cumpliendo con el requerimiento de clasificación (RF01) y delegación (RF02).
    - **Adaptadores por asistente:** actúan como la capa de abstracción técnica, permitiendo la conversión bidireccional entre las interfaces propietarias de cada asistente y el formato unificado del modelo (RF04).
5. **Capa de inferencia:** constituye la base de procesamiento final. Aquí se consolidan el motor semántico y el acceso a modelos locales (Ollama), los cuales proveen la capacidad cognitiva necesaria para la resolución efectiva de las tareas delegadas por los adaptadores.

Bajo este enfoque de capas, la implementación técnica se materializa mediante una arquitectura de microservicios [36]. Esta estructura permite que cada elemento opere de forma independiente y se comunique mediante interfaces estandarizadas. Como se detalla en el diagrama UML de la Figura 3.3, la arquitectura se organiza en tres dimensiones funcionales:

1. **Componente de Orquestación:** es el eje central encargado de la lógica de negocio. Actúa como el único punto de entrada para el usuario a través de una API REST y gestiona la delegación de tareas a través de sus interfaces requeridas (orientadas a la obtención de manifiestos y envío de cargas de trabajo).
2. **Componentes de Adaptación:** cada asistente heterogéneo posee un adaptador dedicado desplegado como un microservicio independiente. Es importante destacar que el diseño es altamente escalable: aunque el diagrama ilustra tres instancias (Física, Matemática y Traductor), el modelo soporta la integración de un número indefinido de asistentes adicionales desplegando nuevos contenedores que respeten el contrato OFP.
3. **Componente de Inferencia Local:** comprende el motor de ejecución Ollama [41], el cual procesa las solicitudes de análisis semántico del orquestador y las peticiones generativas de los asistentes de manera privada. Esto garantiza la soberanía de los datos al ejecutar los LLM de forma estrictamente local.

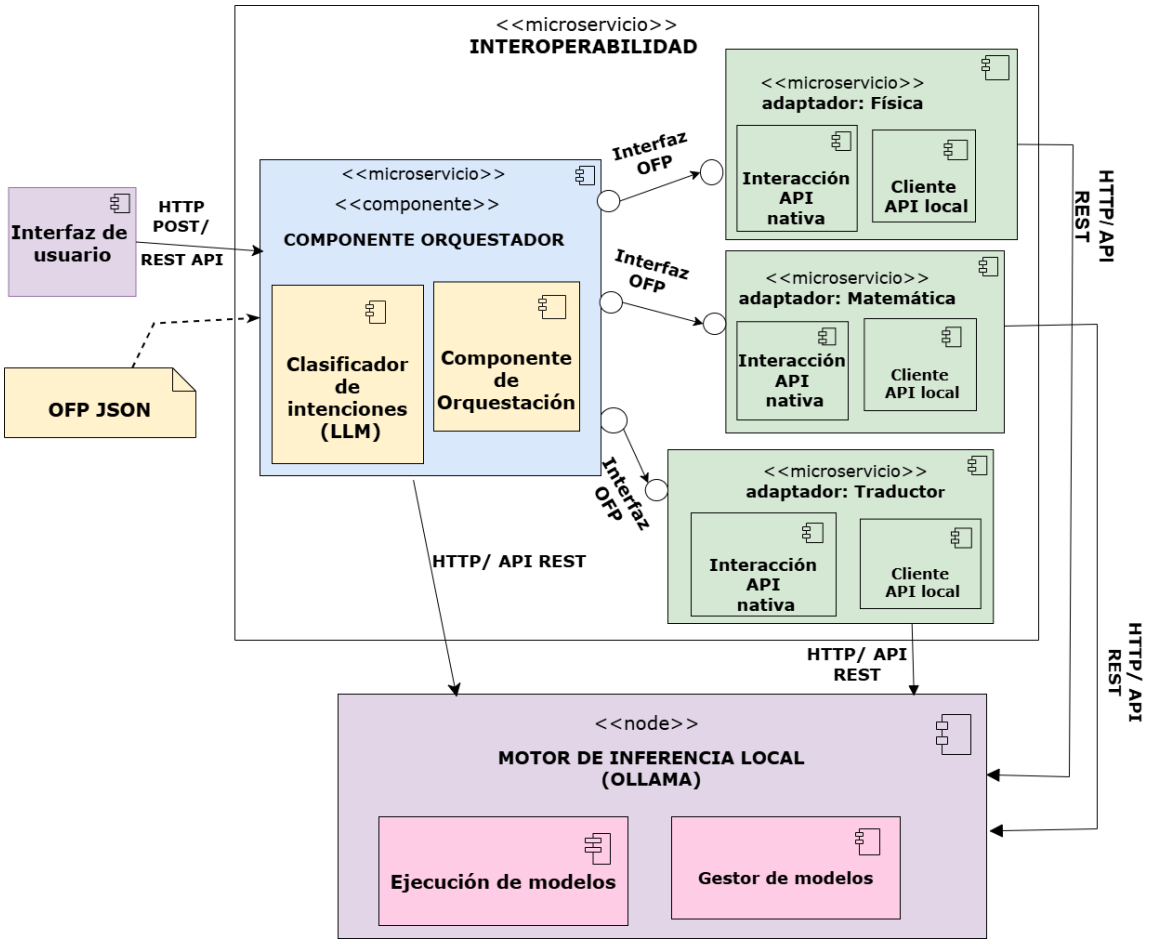


Figura 3.3: Diagrama de componentes UML del modelo de interoperabilidad local.

El orquestador mantiene una filosofía sin estado (*stateless*) y no almacena historiales conversacionales. Sin embargo, los asistentes especializados pueden emplear mecanismos de persistencia semántica mediante *Qdrant* para soportar funcionalidades de recuperación de conocimiento y memoria a largo plazo. Delega la retención cognitiva a la memoria a corto plazo de los mensajes en tránsito, o bien, a las capas internas de los asistentes especializados. Por otro lado, la externalización del cómputo pesado hacia el componente de Inferencia Local asegura que la infraestructura de red del orquestador no se sature, manteniendo la privacidad y soberanía de los datos en el ecosistema institucional [15].

3.3.1 Diseño estructural: patrón Adaptador

Para resolver el requerimiento de bajo acoplamiento y habilitar la coexistencia de asistentes virtuales desarrollados bajo diferentes lenguajes de programación, marcos de trabajo o paradigmas de IA, la arquitectura incorpora el patrón de diseño estructural Adaptador [85].

El objetivo principal de este patrón es adaptar la interfaz de una clase o servicio en otra interfaz esperada por el cliente, permitiendo que componentes con estructuras incompatibles colaboren de forma fluida. En el contexto del modelo propuesto,

el orquestador (que asume el rol de cliente) requiere comunicarse exclusivamente mediante el estándar JSON del protocolo OFP. Por el contrario, los asistentes (que fungen como los componentes Adaptados o *Adaptees*) exponen sus propias API internas, con parámetros y lógicas de respuesta heterogéneas.

La Figura 3.4 presenta el diagrama de clases UML que ilustra la implementación de este patrón en el ecosistema.

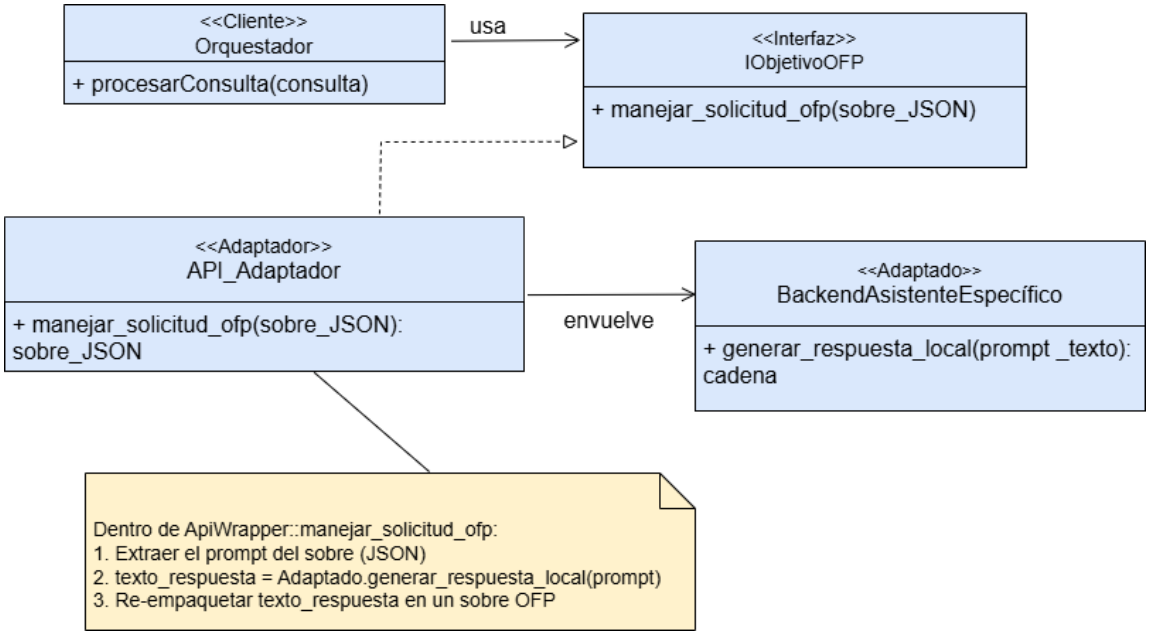


Figura 3.4: Diagrama de clases UML del patrón Adaptador aplicado a los asistentes (Adaptadores o *Wrappers*).

Como se observa en el diagrama, el componente `API_Adaptador` actúa como la clase adaptadora. Su responsabilidad fundamental consiste en cumplir con el contrato establecido por la interfaz `IObjetivoOFP`, lo cual requiere definir un método (e.g., `manejar_solicitud_ofp()`) encargado de recibir, validar y deserializar el sobre OFP entrante. Una vez que se extrae el texto en crudo de la consulta, el adaptador invoca los métodos de procesamiento del `BackendAsistenteEspecífico` (el servicio de respaldo o lógica de dominio).

Al obtener el resultado generado por el asistente, el `API_Adaptador` interviene nuevamente para formatear la respuesta, inyectar el identificador único de la conversación (`id_conversacion`) y reconstruir un sobre OFP válido para devolverlo al orquestador. Esta separación estructural garantiza que, si en el futuro se desea integrar un asistente desarrollado por terceros (e.g., un asistente programado en *Node.js* o una API comercial), el código fuente del orquestador no sufrirá ninguna alteración; bastará únicamente con desarrollar un nuevo microservicio adaptador que respete el contrato de comunicación estandarizado.

3.4 Diseño del protocolo de comunicación y flujo de datos

La piedra angular de la interoperabilidad en el modelo propuesto es el OFP, un estándar basado en JSON-RPC que permite la comunicación uniforme entre el orquestador y los asistentes heterogéneos. Este protocolo define un contrato de mensajería que encapsula la semántica educativa dentro de una estructura de transporte común.

3.4.1 Estructura del mensaje (sobre)

Para garantizar que el orquestador pueda interactuar con cualquier asistente sin conocer su implementación interna, cada comunicación se encapsula en un objeto denominado sobre. Este contenedor incluye los siguientes metadatos críticos:

- **ID de sesión:** identificador único para mantener la trazabilidad de la consulta.
- **Intención:** etiqueta generada por el clasificador para direccionar el mensaje.
- **Payload:** el contenido textual de la consulta del estudiante.
- **Manifest:** información sobre las capacidades del asistente receptor.

3.4.2 Ciclo de vida de la consulta

El flujo de datos sigue un proceso estrictamente secuencial y sin estado, dividido en tres fases principales:

1. **Fase de clasificación:** el orquestador recibe la consulta y utiliza el motor local (Ollama) para determinar a cuál de los asistentes federados (Física, Matemáticas, etc.) corresponde la petición.
2. **Fase de delegación:** una vez identificada la intención, el orquestador envuelve la consulta en el protocolo OFP y la envía al adaptador del asistente correspondiente.
3. **Fase de respuesta:** el asistente procesa la información, genera una respuesta y la devuelve al orquestador, quien finalmente la entrega al usuario final, cerrando el ciclo de comunicación sin persistencia intermedia.

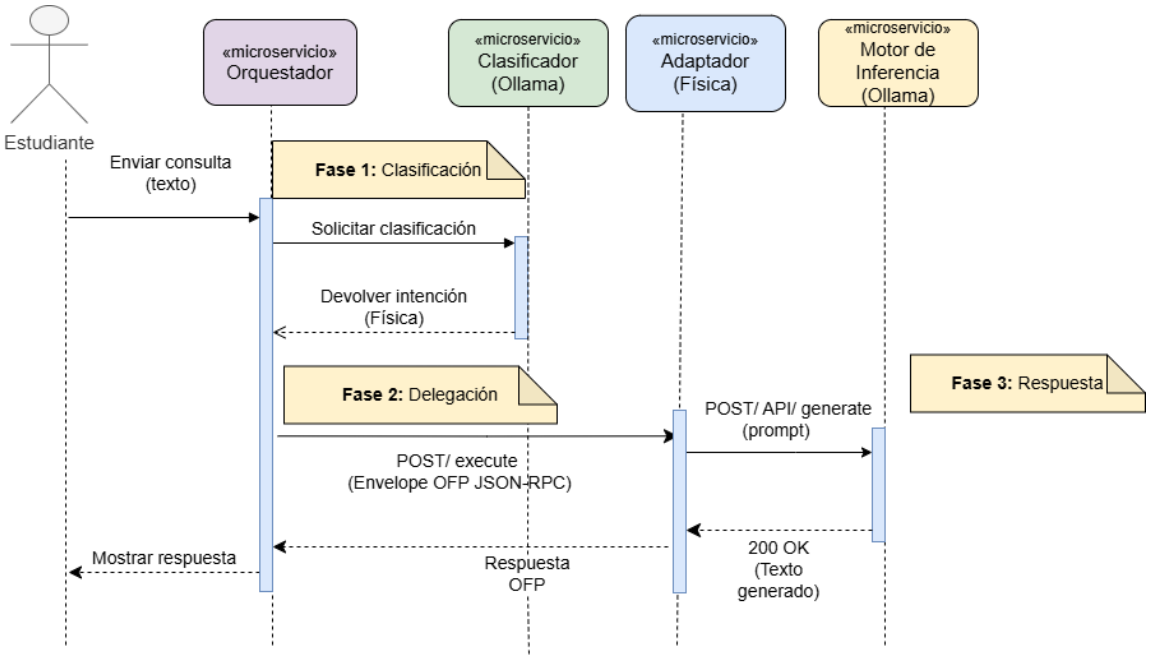


Figura 3.5: Diagrama de secuencia del flujo de mensajes bajo el estándar OFP.

La dinámica de interacción expuesta en el diagrama de secuencia, mostrado en la Figura 3.5 demuestra que el modelo no solo estandariza la comunicación, sino que garantiza el desacoplamiento total entre la lógica de decisión y la generación de contenido pedagógico. Al finalizar este ciclo de vida de la consulta, el sistema queda libre de carga procesal al no requerir persistencia, cumpliendo con los objetivos de eficiencia y privacidad planteados. Una vez definida la arquitectura lógica y el flujo de mensajes, es imperativo detallar los entornos de desarrollo y las herramientas tecnológicas que permitieron materializar esta estructura en un entorno funcional, lo cual se aborda en el siguiente apartado.

3.5 Entorno de desarrollo y herramientas tecnológicas

La implementación del modelo de interoperabilidad requiere un ecosistema tecnológico capaz de soportar la ejecución distribuida de servicios, el procesamiento de lenguaje natural y la comunicación estandarizada entre componentes heterogéneos. En este contexto, se seleccionó un conjunto de herramientas que proporcionan las capacidades necesarias para materializar la arquitectura propuesta y facilitar su despliegue.

A continuación, se describen las tecnologías principales empleadas en el desarrollo del modelo y la justificación técnica de su incorporación dentro del ecosistema:

- **Python 3.11 y FastAPI:** Python se adoptó como lenguaje principal de desarrollo debido a su madurez en el ámbito de la IA y su gestión eficiente de estructuras de datos. Sobre este, se implementó FastAPI, un *framework* moderno basado en Starlette y Pydantic, seleccionado por su elevado rendi-

miento y soporte nativo para programación asíncrona. Estas características son fundamentales para gestionar múltiples solicitudes concurrentes a los asistentes sin bloquear el flujo del orquestador.

- **Virtualización mediante Docker y Docker Compose:** la infraestructura del sistema se implementa mediante contenedores Docker para garantizar el aislamiento. Cada componente se despliega como un servicio independiente, encapsulando sus dependencias. Para la coordinación de los asistentes virtuales se utiliza Docker Compose, herramienta que gestiona la red interna de microservicios y permite reproducir de manera consistente el entorno de ejecución, garantizando la paridad entre desarrollo y producción.
- **Motor de inferencia Ollama:** el procesamiento semántico se realiza mediante Ollama, una plataforma que permite la cuantización y ejecución eficiente de LLM en infraestructura local. Esta decisión estratégica evita depender de API propietarias en la nube, garantizando la privacidad y soberanía de los datos al administrar internamente modelos como Qwen-2.5 y Llama-3.
- **Base de datos vectorial Qdrant:** para habilitar la memoria a largo plazo y RAG, se incorpora Qdrant. Dado que la generación de representaciones vectoriales (*embeddings*) a partir de documentos extensos es computacionalmente costosa, esta base de datos permite indexar y persistir dicha información. Así, los asistentes virtuales pueden recuperar contexto semántico en milisegundos durante la fase de inferencia, sin penalizar la latencia del sistema.

La Tabla 3.4 resume las tecnologías seleccionadas y su justificación principal dentro de la arquitectura:

Tabla 3.4: Resumen de la pila tecnológica del modelo.

Categoría	**Tecnología**	**Justificación Técnica**
Lenguaje	Python 3.11	Desarrollo asíncrono y ecosistema maduro en IA.
Framework API	FastAPI	Exposición de servicios REST de alta concurrencia y baja latencia.
Infraestructura	Docker	Aislamiento de microservicios y eliminación de conflictos de dependencias.
Inferencia LLM	Ollama	Ejecución local eficiente y protección de la soberanía de los datos.
Base de datos vectorial	Qdrant	Indexación persistente para recuperación semántica (RAG) sin impacto en latencia.

La organización de estas tecnologías se representa en la Figura 3.6, donde se observa la distribución por capas de los componentes que conforman el ecosistema. Esta disposición permite separar las responsabilidades asociadas a la interacción con el usuario, la lógica de orquestación, los servicios especializados y los mecanismos de inferencia, proporcionando una estructura modular que facilita la evolución y

mantenimiento del sistema.

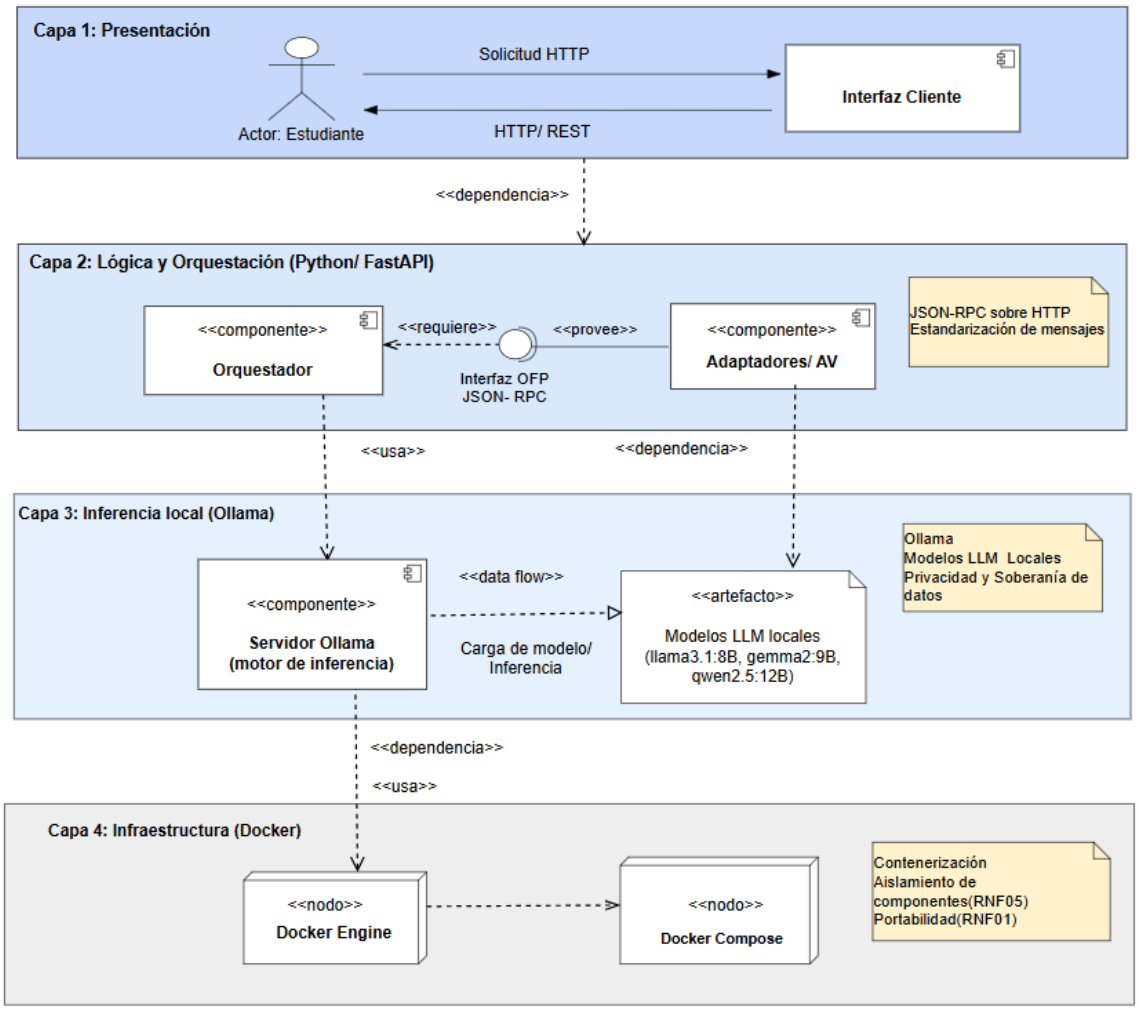


Figura 3.6: Diagrama de capas de la pila tecnológica.

La definición del entorno tecnológico complementa el diseño arquitectónico presentado en las Secciones 3.1, 3.2, 3.3 y 3.4 y proporciona los elementos necesarios para la construcción del prototipo funcional. Con esta base tecnológica establecida, el Capítulo 4 describe la implementación de los componentes del sistema y el proceso de integración de los asistentes dentro del ecosistema de interoperabilidad.

3.6 Trazabilidad del diseño arquitectónico

Para garantizar la integridad del modelo propuesto, resulta fundamental evidenciar la correspondencia entre los requerimientos definidos al inicio de este capítulo y las decisiones arquitectónicas adoptadas. Las Tablas 3.5 y 3.6 presentan la relación entre las necesidades del sistema y los componentes o mecanismos de diseño responsables de satisfacerlas.

Como se observa en la Tabla 3.5, cada requerimiento funcional se encuentra

asociado a uno o más componentes específicos de la arquitectura. A continuación, se describe con mayor detalle la forma en que dichas decisiones de diseño permiten satisfacer las capacidades operativas definidas para el modelo:

Tabla 3.5: Matriz de trazabilidad de requerimientos funcionales.

ID	**Requerimiento**	**Componente arquitectónico asociado**
RF01	Clasificación de intenciones	Orquestador y Motor de Inferencia Local (Ollama)
RF02	Delegación dinámica	Orquestador (motor de decisión y enrutamiento)
RF03	Gestión de contexto	Sobre OFP y transferencia de contexto conversacional
RF04	Traducción de protocolos	Adaptadores basados en el patrón Adaptador
RF05	Paralelismo de ejecución	FastAPI y procesamiento asíncrono de microservicios
RF06	Descubrimiento y registro	API REST de registro y gestión dinámica de manifiestos

La Tabla 3.6 resume la relación entre los atributos de calidad del sistema y los mecanismos arquitectónicos empleados para garantizar su cumplimiento. Seguidamente, se analiza el aporte de cada decisión tecnológica al logro de estos requerimientos:

Tabla 3.6: Matriz de trazabilidad de requerimientos no funcionales.

ID	**Atributo**	**Decisión arquitectónica asociada**
RNF01	Portabilidad	Contenerización mediante Docker y Docker Compose
RNF02	Bajo acoplamiento	Arquitectura por capas, protocolo OFP y patrón Adaptador
RNF03	Latencia	Comunicación local entre microservicios y procesamiento asíncrono
RNF04	Escalabilidad	Arquitectura de microservicios y despliegue independiente de adaptadores
RNF05	Aislamiento	Ejecución de componentes en contenedores independientes
RNF06	Privacidad y soberanía	Inferencia local mediante Ollama y procesamiento interno de datos

Las Tablas 3.5 y 3.6 evidencian la correspondencia directa entre los requerimientos identificados durante la fase de análisis y las decisiones de diseño adoptadas en

la arquitectura propuesta. Esta trazabilidad permite verificar que cada necesidad funcional y atributo de calidad cuenta con mecanismos concretos de implementación dentro del modelo.

La consolidación del diseño arquitectónico, la especificación del ecosistema tecnológico y la validación de su trazabilidad proporcionan la base estructural necesaria para la transición hacia la fase de desarrollo. El análisis detallado en este capítulo establece los fundamentos técnicos requeridos para materializar los conceptos teóricos de interoperabilidad en una solución de software funcional. En el Capítulo 4, se aborda la implementación del modelo, detallando la configuración de la red de microservicios, el desarrollo de la lógica del orquestador y la validación del sistema mediante escenarios de prueba que evalúan la eficacia de la colaboración entre asistentes.

Capítulo 4

Implementación del modelo propuesto

El presente capítulo detalla la transición de los diseños arquitectónicos y conceptuales hacia una implementación técnica funcional, orientada a resolver el desafío de la interoperabilidad en un ecosistema de asistentes virtuales educativos heterogéneos. Durante las siguientes secciones, se documentará la construcción del modelo computacional propuesto, el cual se fundamenta en el protocolo de comunicación OFP. El enfoque principal radica en exponer y justificar las decisiones de Ingeniería de Software a nivel de código fuente, la configuración de la infraestructura mediante técnicas de contenerización, y la gestión del flujo de eventos.

La materialización de este modelo busca consolidar un entorno distribuido, escalable y de bajo acoplamiento [35]. Al detallar el enrutamiento semántico de las consultas y la ejecución paralela o secuencial de los asistentes virtuales de dominio como Matemáticas o Física, se establece la base empírica necesaria para la validación de las hipótesis de la investigación. En este sentido, la implementación no solo refleja la viabilidad técnica del protocolo OFP, sino que también proporciona un marco de referencia reproducible para la integración de herramientas de IA distribuidas en entornos académicos.

4.1 Estructura del proyecto y gestión de dependencias

Para satisfacer de manera estricta el bajo acoplamiento y habilitar la escalabilidad horizontal del ecosistema, la arquitectura del modelo se tradujo en una organización modular basada en el paradigma de microservicios [36]. La gestión técnica de este código fuente se realiza mediante un sistema de control de versiones [86]; sin embargo, en el contexto de este trabajo, la separación estructural de los componentes no requiere repositorios físicamente independientes. En su lugar, todos los módulos coexisten en una misma base de código versionada, pero mantienen un estricto aislamiento lógico mediante configuraciones de entorno (`.env`) dedicadas. Esta estrategia permite administrar de forma independiente las variables, credenciales y parámetros de ejecu-

ción del orquestador, de los asistentes virtuales y de los mecanismos de persistencia, garantizando así que cada componente posea un ciclo de vida de desarrollo, pruebas y despliegue totalmente desacoplado.

4.1.1 Estructura del repositorio

La base de código se estructuró con el objetivo principal de separar físicamente la lógica central de orquestación de las capacidades operativas de cada asistente. Esta división promueve un alto grado de desacoplamiento, aislando el núcleo del modelo —encargado de enrutar los mensajes a través del protocolo OFP— de los asistentes de dominio (el árbol de directorios completo del repositorio se detalla en el Anexo A).

La justificación técnica que sustenta esta topología radica en la separación de responsabilidades. El espacio de trabajo se divide en módulos lógicos: un directorio dedicado a encapsular la lógica individual de cada asistente virtual (`assistants`), incluyendo sus respectivos adaptadores (`api_wrapper.py`) y los documentos base para la RAG [57]. Por su parte, la validación del protocolo de comunicación se centraliza en un módulo independiente (`schemas`), asegurando la adherencia de todos los microservicios al contrato estandarizado en el archivo `conversation-envelope-schema_v1.1.0.json`. Además, para aquellos asistentes virtuales cuya lógica operativa se fundamenta en RAG, se externaliza la persistencia de los espacios vectoriales (`qdrant_storage`). Al mantener las colecciones semánticas de cada disciplina fuera del estado temporal de los contenedores, se evita la re-ejecución del costoso *pipeline* de ingesta documental —el cual abarca la extracción de texto de los archivos PDF, la fragmentación semántica y el cálculo de incrustaciones vectoriales— durante cada ciclo de arranque. En consecuencia, esta estrategia proporciona un mecanismo para que la base de conocimiento persista independientemente de los reinicios del sistema, reduciendo significativamente los tiempos de inicialización y el consumo redundante de recursos de cómputo.

4.1.2 Archivos de configuración e inmutabilidad del entorno

La gestión de dependencias y la configuración de los entornos de ejecución se implementó a nivel de contenedor mediante la plataforma Docker, aislando los procesos del sistema operativo anfitrión [70]. De este modo, cada componente de la arquitectura —ya sea el orquestador o los distintos asistentes de dominio— declara sus dependencias de forma explícita y define las instrucciones para la construcción de su imagen mediante un archivo `Dockerfile` dedicado.

Este enfoque descentralizado es altamente recomendado en arquitecturas de asistentes virtuales heterogéneos, ya que promueve la inmutabilidad del entorno y previene conflictos de versiones entre bibliotecas en tiempo de ejecución (e.g., cuando distintos asistentes requieren versiones mutuamente excluyentes de una misma dependencia).

En el caso específico del módulo `orchestrator`, sus dependencias y versiones, especificadas en el archivo `requirements.txt`, se seleccionaron para satisfacer los requerimientos operativos del núcleo de orquestación. Las bibliotecas `fastapi`, `uvicorn`

y `flask` permiten la exposición de servicios y la gestión de solicitudes HTTP; `jsonschema` se emplea para validar la estructura de los mensajes intercambiados mediante el protocolo OFP, mientras que la integración con el modelo de inferencia local se realiza mediante `ollama`. Por su parte, las bibliotecas `pandas` y `numpy` se utilizan para el procesamiento y análisis de los datos generados durante la evaluación del modelo. El listado completo de dependencias del componente `orchestrator`, correspondiente al archivo `requirements.txt`, se presenta en el Anexo A.

Para asegurar la reproducibilidad de este módulo, su contenedor fue diseñado priorizando la eficiencia en el consumo de recursos. Elegir la imagen base `python:3.10-slim` responde a una decisión de optimización arquitectónica, reduciendo significativamente el tamaño final de la imagen al omitir paquetes innecesarios del sistema operativo base. Asimismo, la secuencia de instrucciones de construcción aprovecha el sistema de caché de capas de Docker: al copiar e instalar primero el archivo de dependencias antes de transferir el resto del código fuente, se evita la reinstalación de bibliotecas cuando únicamente se modifica la lógica algorítmica de `orchestrator.py`.

Como medidas preventivas de estabilidad, se configuraron tiempos de espera extendidos para la descarga de paquetes pesados, mitigando posibles fallos por intermitencias de red. Finalmente, el contenedor expone el puerto de escucha (`5000`) estrictamente necesario para habilitar la comunicación asíncrona dentro de la red interna (la especificación completa del `Dockerfile` se detalla en el Anexo B). Al estandarizar la provisión de la infraestructura a través de estas definiciones, se facilita la reproducibilidad del modelo completo en cualquier entorno de evaluación.

4.2 Configuración de la infraestructura y orquestación

Una vez definida la organización del repositorio y el empaquetado de los componentes, la integración del modelo requiere un entorno de infraestructura que coordine la ejecución simultánea de los servicios. Para este propósito, se seleccionó Docker Compose como herramienta de orquestación ligera, lo que permite consolidar la topología de red, el almacenamiento persistente y el aprovisionamiento de recursos del sistema bajo un único manifiesto declarativo estructurado (la especificación completa del archivo `docker-compose.yml` se detalla en el Anexo B).

Este enfoque centralizado simplifica el despliegue de una arquitectura distribuida compleja, coordinando de forma unificada el motor de orquestación, la base de datos vectorial (`qdrant`), el servidor de inferencia de modelos de lenguaje (`ollama`) y los contenedores acoplados para cada asistente de dominio específico.

4.2.1 Red de contenedores y comunicación interna

El flujo de mensajes determinado por el protocolo OFP requiere canales de comunicación fiables y lógicamente aislados. Para cumplir con estas especificaciones operativas, se implementó una red virtual dedicada utilizando el controlador de puente (*bridge driver*) nativo de la plataforma de contenerización.

Esta abstracción de red habilita el servicio de resolución de nombres interno. En lugar de depender de direcciones IP estáticas que comprometerían la portabilidad y escalabilidad del modelo, los contenedores resuelven la ubicación de sus dependencias utilizando el nombre del servicio como identificador de red (*hostname*). Un patrón representativo de este diseño se observa en la inyección de variables de entorno para la comunicación interna entre los componentes *adapters* y sus respectivos *backends* lógicos. Mediante este mecanismo, se parametriza la ruta de acceso de forma dinámica, asociando explícitamente las dependencias de arranque del ciclo de vida del contenedor a través de directivas de orden de inicialización (`depends_on`).

Asimismo, al mapear únicamente puertos específicos hacia el sistema anfitrión (como el puerto `5000` para el orquestador principal o el puerto `6333` para el motor de búsqueda vectorial) y mantener el resto del tráfico encapsulado en la red virtual dedicada, se reduce la superficie de exposición del sistema. Esta decisión de diseño promueve que la comunicación inter-asistente y las consultas de persistencia transcurran en un entorno de red lógicamente aislado del exterior.

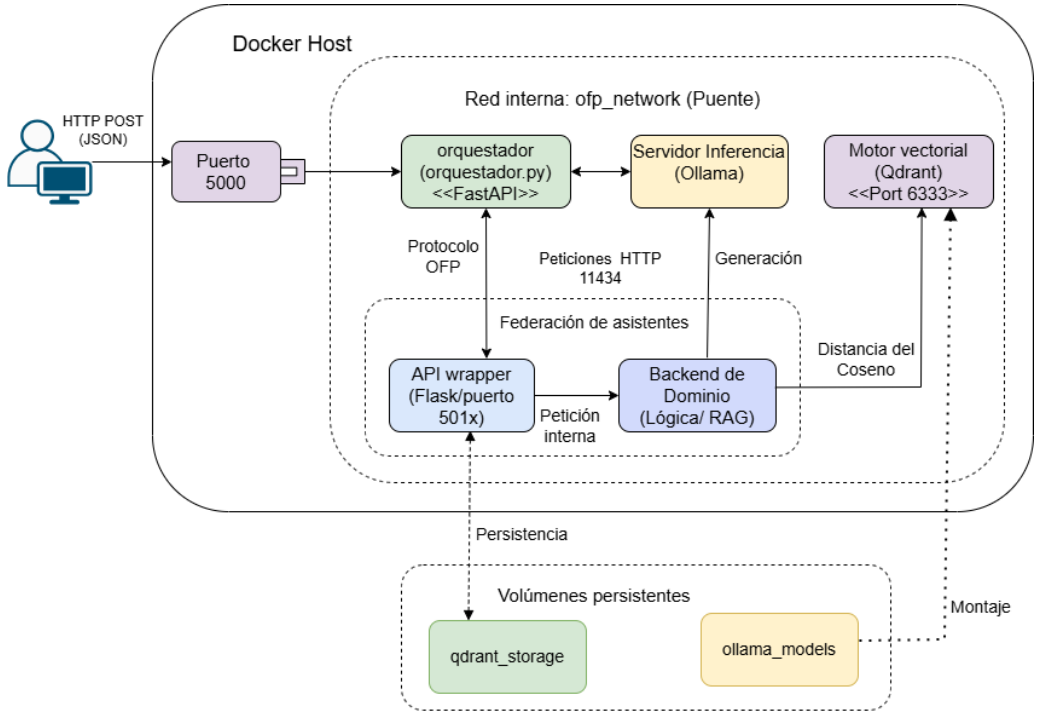


Figura 4.1: Arquitectura de despliegue del ecosistema, ilustrando la topología de la red virtual `ofp_network`, la federación de asistentes y el montaje de volúmenes persistentes.

4.2.2 Asignación de recursos de hardware y gestión de concurrencia

Dado que el núcleo de procesamiento semántico del modelo se fundamenta en la ejecución local de modelos LLM a través del servicio `ollama`, resulta fundamental establecer políticas de configuración rigurosas para la asignación de hardware y el control de concurrencia de peticiones en un entorno con recursos compartidos [82, 81].

La configuración del motor de inferencia se apoya en variables de entorno diseñadas para mitigar el riesgo de saturación de memoria y optimizar los tiempos de latencia del ecosistema. En primer lugar, la directiva que limita la cantidad de capas de la red neuronal delegadas a la aceleración por hardware permite balancear la carga computacional. Al evitar forzar la transferencia total de los parámetros del modelo hacia la memoria VRAM, se contribuye a la estabilidad del sistema operativo anfitrión, permitiendo que las capas restantes se distribuyan en los hilos de procesamiento de la CPU.

Por otro lado, se configuró el parámetro de persistencia en memoria del modelo de forma indefinida, eliminando los retardos temporales asociados a la inicialización y desasignación de tensores en el hardware gráfico entre consultas secuenciales.

Finalmente, se configuraron mecanismos de control de concurrencia para regular el acceso al motor de inferencia como se muestra en el Anexo B. En particular, se limitó la carga simultánea a un único modelo en memoria y la ejecución a una sola inferencia concurrente, estableciendo además una cola de espera para las solicitudes adicionales. Esta configuración contribuye a mantener un comportamiento estable del sistema, evitando la sobrecarga de la memoria disponible y reduciendo el riesgo de fallos cuando múltiples asistentes virtuales realizan solicitudes de inferencia de manera simultánea.

4.3 Desarrollo del modelo de datos basado en OFP

Para promover la interoperabilidad entre los múltiples asistentes virtuales heterogéneos y el núcleo de orquestación, fue necesario establecer un contrato de comunicación estandarizado. El modelo propuesto adopta e implementa el OFP, un protocolo de mensajería basado en el formato de intercambio de datos JSON. Este enfoque mitiga la ambigüedad en la transmisión de información y define una semántica común para la interacción bidireccional entre los componentes distribuidos. Con el propósito de ofrecer una visión introductoria antes de detallar la especificación técnica del protocolo, la Figura 4.2 ilustra de manera esquemática el flujo básico de comunicación mediante OFP dentro de la arquitectura propuesta. En dicho diagrama convergen tres actores principales: el usuario, que expone la capa de consumo externa mediante peticiones HTTP en formato JSON; el orquestador, encargado del enrutamiento semántico de la consulta y de la fase de síntesis de la respuesta final; y el asistente virtual, compuesto internamente por un adaptador (*wrapper*) que traduce el sobre OFP hacia el motor de inferencia.

Como se aprecia en la figura, la comunicación entre el orquestador y cada asistente virtual se encuentra íntegramente encapsulada bajo el protocolo OFP, materializándose principalmente a través del evento `utterance`, tanto para la propagación de la consulta original como para el retorno de la respuesta generada por el modelo de lenguaje. De manera complementaria, el evento `publishManifest` —representado mediante una flecha discontinua— ilustra el mecanismo de registro dinámico que cada asistente ejecuta al inicializar su ciclo de vida, permitiendo su incorporación autónoma al ecosistema sin intervención manual.

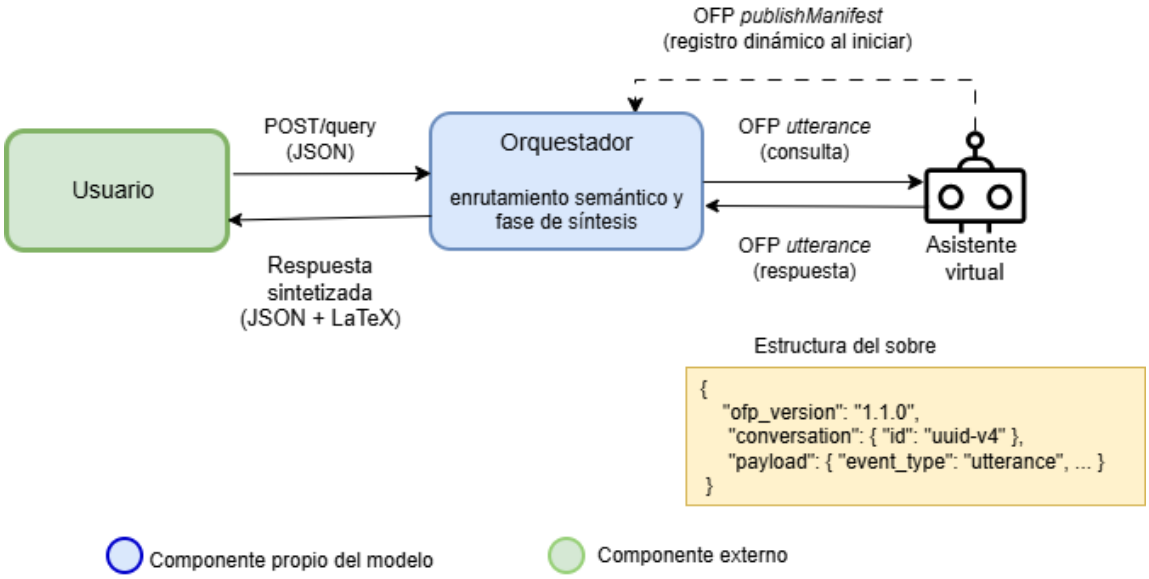


Figura 4.2: Flujo básico de comunicación mediante el protocolo OFP entre el usuario, el orquestador y un asistente virtual, distinguiendo los componentes desarrollados en esta tesis y componentes externos

4.3.1 Estructura base del sobre y control de sesión

El diseño del protocolo OFP se fundamenta en el patrón Mensajería de Sobre (*Envelope Wrapper*) [69]. Cada mensaje transmitido a través de la red de contenedores se encapsula en una estructura base inmutable que contiene los metadatos de enrutamiento, separando lógicamente la capa de transporte de la carga útil (*payload*).

A nivel operativo, el orquestador fue diseñado bajo el principio de ausencia de estado. Esto significa que no depende de bases de datos relacionales ni de cachés en memoria para recordar el historial de interacciones activas. En su lugar, el control de sesión se delega completamente al modelo de datos mediante el atributo `conversation:id` (la estructura base del sobre OFP se ilustra en el Anexo C).

```
{
  "ofp_version": "1.1.0",
  "conversation": { "id": "f47ac10b-58cc-4372-a567-0e02b2c3d479" },
  "payload": { ... }
}
```

El identificador de conversación utiliza el estándar UUID versión 4 [87]. Cuando una petición de usuario ingresa al sistema, el orquestador asigna o lee este identificador y lo inyecta en el sobre. Cada asistente que recibe el mensaje tiene la obligación de incluir el mismo identificador en su respuesta. Esta trazabilidad determinista permite que el orquestador correlacione múltiples respuestas asíncronas procedentes de distintos asistentes matemáticos o físicos y las ensamble de manera coherente antes de devolver el resultado final, facilitando la resolución de la concurrencia sin incrementar de forma insostenible el consumo de memoria del servidor.

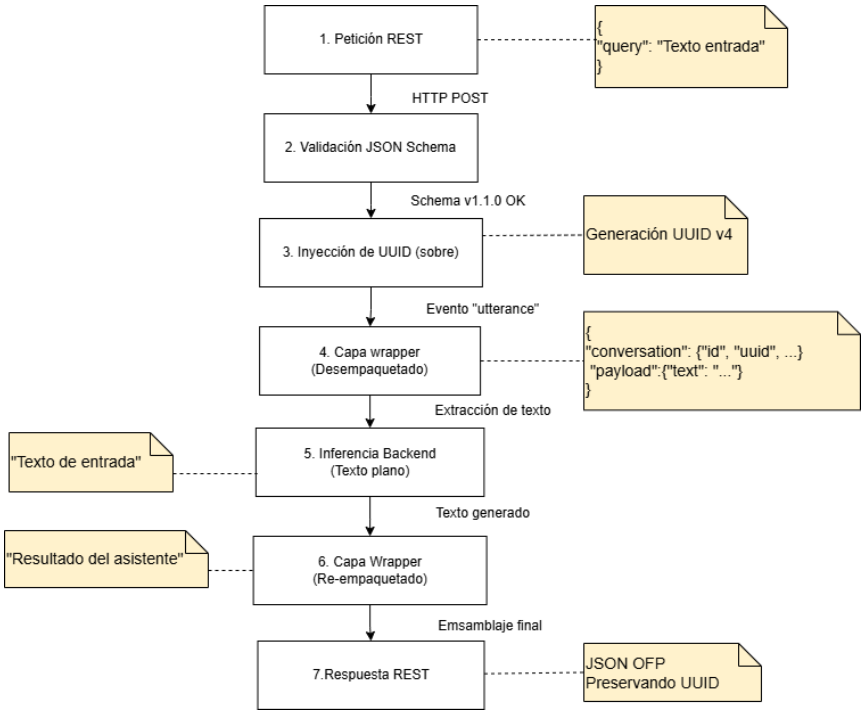


Figura 4.3: Diagrama de flujo del ciclo de vida de una consulta: encapsulamiento, cambio de estado y persistencia del identificador de sesión a través del protocolo OFP.

4.3.2 Gestión de eventos (*utterance* y *publishManifest*)

Dentro del sobre, el atributo `payload` define la naturaleza de la interacción. El ecosistema opera principalmente mediante una arquitectura orientada a eventos [88], donde destacan dos eventos fundamentales para el ciclo de vida del modelo: `utterance` y `publishManifest`.

El evento `utterance` (enunciado o emisión) es el vehículo primario para el intercambio de lenguaje natural. Se utiliza tanto para transmitir la consulta original del usuario hacia los asistentes, como para devolver el texto generado por un LLM. Su estructura interna es intencionalmente minimalista para reducir la sobrecarga de red (ver Anexo C).

```
"payload": {
  "event_type": "utterance",
  "content": { "text": "Explica la diferencia entre memoria
     RAM y ROM." }
}
```

Por otro lado, el evento `publishManifest` es el mecanismo que habilita la extensibilidad dinámica del ecosistema. Cuando un nuevo contenedor se inicializa (e.g., `assistant_computacion`), este emite un evento `publishManifest` hacia el orquestador. La carga útil de este evento contiene un arreglo de descriptores semánticos y capacidades técnicas del asistente. Este registro dinámico previene la codificación rígida (*hardcoding*) de las rutas en el orquestador; si en el futuro se añade un nuevo asistente virtual, este solo requiere publicar su manifiesto para ser inmediatamente reconocido y enrutado por el orquestador principal.

4.3.3 Validación estructural mediante JSON Schema

Dada la naturaleza distribuida del modelo, un paquete malformado proveniente de un asistente podría provocar excepciones no controladas en el núcleo de enrutamiento. Para mitigar esta vulnerabilidad, se implementó un mecanismo de validación estructural estricta previa a cualquier procesamiento lógico, empleando la especificación JSON Schema [89].

Como se documentó en la organización del repositorio, el archivo central de validación es `conversation-envelope-schema_v1.1.0.json`. Este documento define de forma declarativa los tipos de datos esperados, las propiedades obligatorias y las restricciones de formato algorítmico (el esquema de validación implementado se detalla en el Anexo C).

Durante la ejecución, el orquestador y los *wrappers* de cada asistente virtual emplean la biblioteca validadora para contrastar cada mensaje entrante contra este contrato. Si un paquete no contiene un UUID válido o presenta una versión no soportada, la capa de red lo descarta inmediatamente y retorna un código de error HTTP 400 (*Bad Request*). Esta barrera de validación determinista asegura que los recursos computacionales de la CPU y la GPU, respectivamente, no se desperdicien procesando estructuras de datos corruptas, incrementando significativamente la tolerancia a fallos del modelo computacional.

4.4 Implementación del orquestador

El orquestador actúa como el núcleo cognitivo y el enrutador de tráfico principal del modelo. A diferencia de los asistentes periféricos, su diseño arquitectónico lo exime de poseer conocimiento de dominio específico; su responsabilidad exclusiva es analizar la intención del usuario, determinar la ruta de ejecución óptima entre los asistentes virtuales disponibles, coordinar las peticiones a través de la red y sintetizar una respuesta coherente [5]. Para cumplir con estas responsabilidades en tiempo real, el orquestador implementa técnicas avanzadas de procesamiento de lenguaje natural y concurrencia mediante hilos.

4.4.1 Clasificación semántica mediante *prompt engineering*

El primer desafío computacional del orquestador es el enrutamiento semántico: dado un enunciado (*utterance*) en lenguaje natural, el sistema debe inferir qué asistente virtual o combinación de asistentes virtuales está mejor capacitado para resolver la solicitud. En lugar de depender de heurísticas basadas en coincidencia de palabras clave —las cuales resultan frágiles ante la ambigüedad del lenguaje—, se implementó un mecanismo de clasificación dinámica soportado por un modelo LLM a través del servicio local de Ollama.

El proceso de clasificación implementado por el orquestador se apoya en técnicas de *prompt engineering*, las cuales permiten orientar el comportamiento del modelo de lenguaje para generar una salida estructurada y consistente [90]. En particular, se emplea un *prompt* determinista que rige el enrutamiento, la exclusión mutua de tareas

y la propagación de contexto entre asistentes, cuyo código fuente detallado se presenta en el Anexo D. Este enfoque permite que el orquestador adapte dinámicamente el proceso de clasificación al estado actual del ecosistema mediante la incorporación de los manifiestos de los asistentes disponibles. Asimismo, al instruir al modelo para restringir su salida a un formato JSON predefinido, se facilita el procesamiento y la deserialización automática de la respuesta, como se ilustra en la siguiente salida esperada.

```
["physics-assistant", "algebra-backend"]
```

La definición explícita del formato de salida permite que el orquestador procese la respuesta del LLM mediante su deserialización en formato JSON, facilitando la selección e invocación automática de los asistentes virtuales correspondientes.

4.4.2 Estrategia paralela y secuencial

Una vez que el enrutador semántico determina los asistentes virtuales objetivo, el orquestador debe gestionar las peticiones HTTP a través del protocolo OFP. Dado que la inferencia local y la búsqueda vectorial (RAG) en los asistentes introducen una latencia inherente debido a las operaciones de entrada/salida, invocar a los asistentes de forma estrictamente secuencial acumularía la latencia de todos ellos, saturando el tiempo de respuesta percibido por el usuario.

Para solventar esta limitación, se implementó concurrencia mediante hilos del sistema operativo (*multithreading*), despachando una petición HTTP independiente por cada asistente dentro de una misma etapa y sincronizando su finalización antes de avanzar a la etapa siguiente. El modelo evalúa el grafo de dependencias de la consulta para determinar la estrategia de ejecución óptima (el código de implementación de esta rutina se documenta en el Anexo D):

- **Ejecución paralela:** cuando la clasificación semántica determina que la consulta requiere asistentes virtuales independientes (e.g., evaluar el contexto histórico y la gramática de un texto simultáneamente mediante los asistentes de Historia y Español), el orquestador despacha las peticiones de forma concurrente. La multiplexación de estas tareas asegura que el tiempo total de espera de red sea equivalente al del asistente con mayor latencia, en lugar de la suma secuencial de todos, reduciendo drásticamente el tiempo de respuesta percibido por el usuario.
- **Ejecución secuencial:** en escenarios donde existe dependencia de contexto (e.g., una solicitud de traducción cuyo resultado debe ser procesado posteriormente por el motor matemático avanzado), el orquestador muta su comportamiento y ejecuta las llamadas en serie, inyectando dinámicamente la salida del primer asistente como contexto en el sobre del segundo.

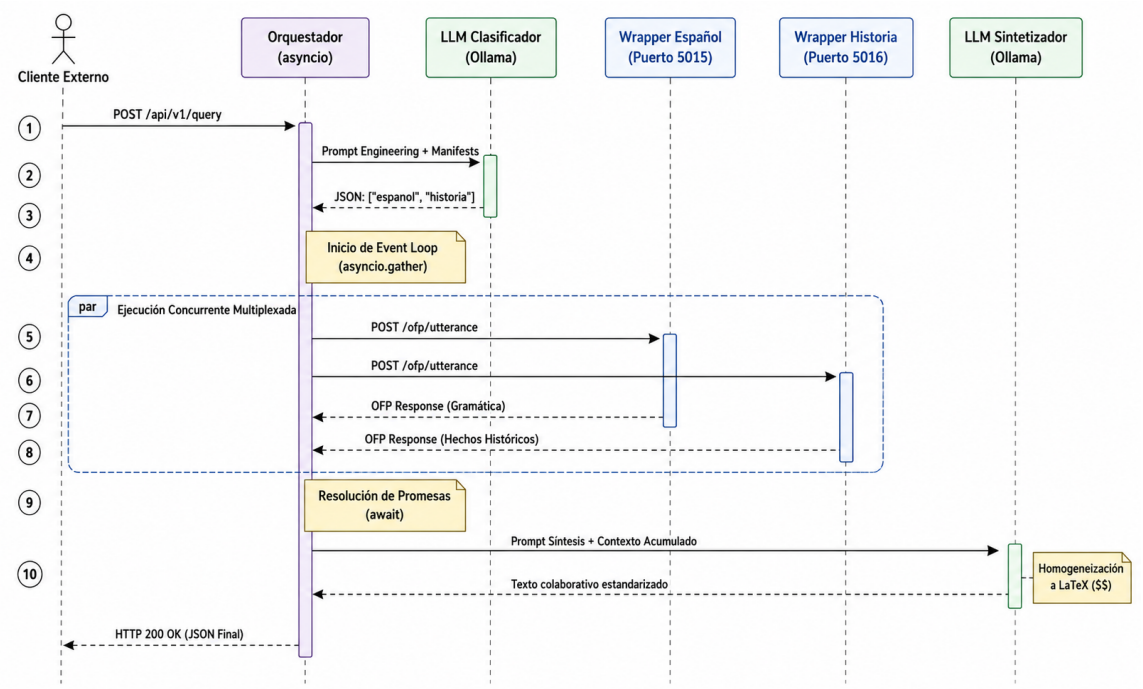


Figura 4.4: Diagrama de secuencia del orquestador, peticiones hacia los asistentes de dominio y la posterior fase de síntesis para la estandarización tipográfica.

4.4.3 Inyección dinámica de reglas y homogeneización de formato

El último paso en el ciclo de vida de la petición dentro del orquestador es la resolución de conflictos y la homogeneización. Cuando se obtienen múltiples respuestas de distintos asistentes, es altamente probable que difieran en tono, estilo o estructura. Entregar estas respuestas crudas concatenadas afectaría negativamente la fluidez de la experiencia pedagógica.

Para resolver esto, el orquestador emplea un segundo pase de inferencia LLM denominado "Fase de Síntesis". En esta etapa, las respuestas generadas por los asistentes se agrupan y se transfieren al modelo cognitivo junto con un conjunto de directivas inyectadas dinámicamente. El objetivo principal no es alterar la precisión fáctica de los asistentes periféricos (como las ecuaciones derivadas por el asistente de Física), sino estandarizar su presentación visual y narrativa (ver Anexo D para la plantilla de síntesis).

Esta inyección de reglas de formato resulta crítica para el acoplamiento con la interfaz de usuario. Al instruir explícitamente al orquestador para que normalice la notación matemática hacia el estándar LaTeX, compatible con motores de renderizado como KaTeX, se libera a los asistentes virtuales individuales de la responsabilidad del formato de presentación. De este modo, se consolida el principio de "separación de responsabilidades" [91], donde los asistentes virtuales periféricos se limitan estrictamente a la resolución cognitiva del problema, mientras que el orquestador provee la integridad estructural de la salida entregada.

4.5 Desarrollo de adaptadores y memoria a largo plazo

La arquitectura distribuida del modelo plantea un doble desafío: por un lado, se requiere estandarizar la comunicación entre entidades tecnológicamente heterogéneas; por otro, es necesario dotar a los asistentes encargados de dominios factuales de una base de conocimiento fundamentada para reducir la incidencia de alucinaciones asociadas a los LLM. Esta sección detalla la implementación del patrón de diseño que resuelve la interoperabilidad de red, así como el procesamiento de datos que conforma la memoria semántica para aquellos asistentes que lo requieren.

4.5.1 Construcción de los adaptadores

El ecosistema está compuesto por asistentes con lógicas de inferencia diversas: algunos consultan directamente a un LLM, otros utilizan motores basados en reglas algorítmicas y un tercer grupo emplea recuperación de documentos. Para alinearse con el principio de responsabilidad única, esta heterogeneidad interna debe ocultarse de la capa de red encargada de la validación y el enrutamiento del protocolo OFP.

Para lograr esto, se implementó el patrón de diseño estructural Adaptador (*Adapter Pattern*) [85] mediante componentes denominados *wrappers*. Cada asistente en el ecosistema cuenta con un contenedor intermedio (`api_wrapper.py`) que actúa como una capa de traducción entre el orquestador y el *backend* lógico del asistente.

La función de este componente es interceptar el sobre JSON estándar, deserializarlo, extraer el texto plano de la consulta y enviarlo al motor interno, independientemente del paradigma de IA que este último utilice. Una vez que el *backend* genera la respuesta, el adaptador la re-empaqueta en un formato OFP válido y la devuelve al orquestador, preservando intacto el identificador de sesión (`conversation:id`). La implementación técnica de este mecanismo de traducción, apoyada en el marco de trabajo Flask, se detalla en el Anexo E.

Esta abstracción metodológica promueve que el orquestador se mantenga agnóstico respecto a la implementación interna de cada módulo. En consecuencia, permite que desarrolladores futuros integren asistentes virtuales construidos en lenguajes de programación o *frameworks* alternativos sin necesidad de modificar el núcleo del orquestador principal, siempre que provean un adaptador compatible con el contrato del protocolo OFP.

4.5.2 Arquitectura RAG: ingesta documental y recuperación de contexto

Es importante precisar que la arquitectura del ecosistema admite múltiples paradigmas cognitivos simultáneamente. Algunos asistentes operativos, como los dedicados a la traducción o a la resolución matemática, operan de manera autónoma consultando directamente al modelo generativo. Por el contrario, los asistentes asignados a dominios teóricos complejos, como Historia, Química y Física, requieren investigar en

literatura académica externa para fundamentar sus respuestas.

Dado que la ventana de contexto de los LLM locales es limitada por las restricciones de hardware, no es computacionalmente factible inyectar corpus bibliográficos enteros en cada consulta. Para este subconjunto de asistentes, la solución implementada consistió en la arquitectura RAG [57], la cual inicia con un *pipeline* de ingesta de datos secuencial que transforma documentos no estructurados en representaciones matemáticas de alta dimensionalidad [92].

La primera fase de este proceso inicia con la extracción de texto crudo proveniente de los archivos PDF, los cuales son preseleccionados y curados por los expertos de dominio para garantizar la calidad de la fuente. Posteriormente, en la segunda fase, el texto extraído se somete a una fragmentación semántica (*chunking*). En lugar de dividir el corpus arbitrariamente por longitud estricta de caracteres, se aplicaron estrategias de solapamiento (*overlapping*). Esta técnica mitiga la pérdida del hilo conductor y preserva el contexto semántico entre párrafos consecutivos.

En la tercera fase, los bloques de texto fragmentados se procesan a través de un modelo de incrustaciones (*embedding model*) que proyecta la semántica de la información en un espacio vectorial. Finalmente, estos vectores resultantes se almacenan de forma persistente en *Qdrant*, un motor de base de datos vectorial de alto rendimiento configurado en la red interna (el código de inicialización de estas colecciones se encuentra documentado en el Anexo E). La selección de la métrica de Similitud del Coseno en esta etapa resulta fundamental, ya que permite calcular la similitud angular entre los vectores, optimizando la precisión algorítmica de la base de datos [93].

Una vez consolidada esta base documental, la culminación de la memoria semántica a largo plazo se materializa en la fase de recuperación, la cual interviene directamente en el flujo de ejecución del *backend* interno del asistente, aislada lógicamente de la red OFP.

Cuando un usuario interactúa con un asistente basado en RAG, la consulta original se vectoriza utilizando el mismo modelo de incrustaciones empleado en la fase de ingesta. Este vector resultante se transmite al motor Qdrant, el cual ejecuta una búsqueda algorítmica de los k-vecinos más cercanos (*k-Nearest Neighbors*) dentro del espacio latente de la colección del dominio correspondiente.

Los bloques de texto recuperados representan la evidencia literaria empírica. Posteriormente, esta información se concatena dinámicamente mediante inyección de contexto en el *prompt* del modelo de lenguaje. Esta directiva estructural obliga al algoritmo generativo a fundamentar su respuesta exclusivamente en los hechos estipulados en la referencia académica (ver Anexo E para la estructura de la plantilla instruccional).

Esta metodología híbrida provee un balance de eficiencia en el modelo global: los asistentes que no requieren contexto histórico responden de manera autónoma con baja latencia, mientras que la integración RAG en los dominios teóricos mitiga significativamente la variabilidad estocástica y las imprecisiones. Al estandarizar este flujo selectivo, se favorece que la calidad pedagógica del modelo dependa de herramientas especializadas acordes a la naturaleza de cada disciplina, satisfaciendo los estándares de rigor exigidos en un entorno educativo formal.

4.6 Interfaz de consumo y estandarización de salida

Aunque el modelo propuesto se centra metodológicamente en la orquestación y la interoperabilidad de asistentes virtuales distribuidos, la interacción empírica con el sistema requiere una capa de entrada y salida de datos estandarizada. En lugar de acoplar el núcleo lógico a una interfaz gráfica de usuario convencional, se implementó una interfaz de consumo basada en la arquitectura de servicios web REST [68], la cual actúa como punto de acceso unificado para clientes externos o herramientas de evaluación automatizadas.

4.6.1 Diseño de la capa de presentación (API REST)

La capa de presentación del modelo se materializa a través de un punto de enlace de red, diseñado para recibir peticiones síncronas estructuradas en formato JSON. Esta decisión técnica responde a la necesidad de mantener el sistema agnóstico a la plataforma de visualización, permitiendo que el ecosistema pueda ser integrado sin modificaciones en aplicaciones web, dispositivos móviles o herramientas de línea de comandos.

El acceso al sistema se realiza mediante el método HTTP POST dirigido al orquestador principal. La estructura de la consulta (*query*) encapsula únicamente la intención del usuario en lenguaje natural, delegando al modelo toda la complejidad del enrutamiento semántico y la síntesis colaborativa detallada en secciones anteriores (un ejemplo de la estructura de estas peticiones para problemas multidisciplinarios se documenta en el Anexo F).

Esta interfaz abstracta permite evaluar la capacidad del orquestador para fragmentar consultas complejas y delegar tareas concurrentes (e.g., asignar el cálculo de fuerzas al asistente de Física, la derivación al asistente de Matemáticas y la conversión idiomática al asistente de traducción). La respuesta entregada por el modelo omite el texto plano desestructurado en favor de un objeto JSON que preserva la trazabilidad de la conversación y la integridad de los datos procesados.

4.6.2 Estandarización de respuestas y tipografía matemática

Uno de los desafíos técnicos en la visualización de salidas provenientes de LLM heterogéneos es la inconsistencia estructural en la representación de notación científica y simbología matemática. Para resolver esta divergencia, el orquestador implementa una capa de "normalización de salida", la cual instruye al modelo de síntesis a formatear todas las entidades matemáticas bajo el estándar tipográfico de LaTeX [94].

Esta normalización resulta fundamental para habilitar el renderizado posterior mediante bibliotecas de alto rendimiento en el lado del cliente (*frontend*), tales como *KaTeX* o *MathJax*. El modelo instruye dinámicamente a los asistentes para que delimiten toda expresión matemática utilizando los marcadores estándar (doble símbolo de dólar para bloques y uno sencillo para fórmulas en línea). El impacto de esta regla se refleja en la cadena de texto de la respuesta final:

```
"response": "La fuerza se calcula como: $$F = m \cdot a =
   18000 \text{ N}$$"
```

La justificación arquitectónica de este enfoque radica en el principio de "separación de responsabilidades". El modelo computacional desarrollado no se responsabiliza del renderizado visual final de las ecuaciones, sino de proveer integridad y garantía estructural en la capa de datos. Al asegurar que la salida del orquestador cumpla con el estándar declarativo de LaTeX, se facilita que cualquier interfaz de usuario que consuma esta API pueda visualizar las expresiones con calidad tipográfica profesional, preservando el rigor visual exigido en las disciplinas de ciencias exactas e ingeniería.

Finalmente, esta sección muestra la capacidad del modelo para consolidar una respuesta colaborativa coherente. Al unificar los resultados de múltiples especialistas en un flujo de datos estructurado, el ecosistema transita de ser una colección aislada de asistentes virtuales conversacionales hacia una arquitectura integrada para la resolución de problemas multidisciplinarios.

4.7 Mecanismo de extensibilidad del ecosistema

Una de las limitaciones más críticas en las arquitecturas monolíticas de IA es su rigidez ante la evolución del dominio de conocimiento. En el contexto educativo, los planes de estudio y las disciplinas cambian constantemente, por lo que la arquitectura del modelo propuesto se diseñó bajo el principio de apertura/clausura de la metodología SOLID [95]: el sistema se estructura de forma abierta a la extensión, pero cerrado a la modificación de su núcleo de enrutamiento.

Esta característica de diseño se materializa mediante la federación de asistente, un patrón arquitectónico que permite integrar nuevos módulos cognitivos sin alterar el código fuente del orquestador [23]. La federación descentraliza la base de conocimiento, promoviendo que cada nuevo asistente sea desarrollado, entrenado y empaquetado de manera autónoma, para posteriormente incorporarse al ecosistema como un nodo cooperativo desacoplado.

4.7.1 Protocolo de registro dinámico en caliente

Para mitigar los tiempos de inactividad y evitar la reconfiguración manual de rutas estáticas, la incorporación de un nuevo asistente se gestiona mediante un mecanismo de descubrimiento de servicios [96], denominado en este modelo como "registro dinámico en caliente".

Este proceso se ejecuta a través de la red interna de contenedores. Cuando un nuevo asistente (e.g., un asistente orientado a Biología) completa su secuencia de arranque, su respectivo componente *wrapper* emite automáticamente un evento OFP de tipo `publishManifest` hacia el punto de enlace de registro del orquestador. La estructura estandarizada de la carga útil de este manifiesto se documenta detalladamente en el Anexo G.

Al recibir este paquete, el orquestador valida su integridad estructural mediante el esquema JSON correspondiente. Si la validación resulta exitosa, el orquestador

actualiza dinámicamente su tabla de enrutamiento en memoria. Dicha estructura de datos es consultada por el modelo de lenguaje clasificador durante la fase de análisis semántico, permitiendo que el orquestador incluya de manera inmediata al nuevo experto disciplinar en su flujo de toma de decisiones.

La principal ventaja arquitectónica de este enfoque radica en la preservación del bajo acoplamiento: el orquestador no requiere un conocimiento axiomático previo sobre la cantidad o la naturaleza de los asistentes que conformarán el entorno. El registro en caliente facilita que el ecosistema escale horizontalmente de forma flexible, transformando un canal estático de mensajes en un sistema multiasistente verdaderamente dinámico y autoconfigurable.

4.8 Instrumentación del sistema para trazabilidad y evaluación

Para dar cumplimiento al objetivo específico referente a la evaluación del desempeño del modelo, fue necesario dotar a la arquitectura de capacidades nativas de observabilidad. En sistemas distribuidos basados en LLM, las pruebas unitarias tradicionales resultan insuficientes debido a la naturaleza estocástica de las respuestas. Por ello, se instrumentó un módulo automatizado para habilitar una futura evaluación semántica y computacional, extrayendo métricas de fiabilidad y rendimiento directamente de la red de contenedores [97].

El diseño de esta instrumentación soporta dos niveles de auditoría: la prueba aislada de los asistentes periféricos (fase 1) y la recolección de trazabilidad integral del ecosistema completo (fase 2).

4.8.1 Interfaces para la evaluación individual de asistentes

Para permitir el aislamiento del rendimiento cognitivo de cada asistente (excluyendo la latencia y las decisiones de enrutamiento del orquestador),el módulo de evaluación genera dinámicamente cargas útiles (*payloads*) bajo el protocolo OFP y las inyecta directamente en el adaptador (*wrapper*) de cada asistente.

Para auditar la exactitud semántica frente a un documento de referencia, el módulo implementa rutinas algorítmicas de vectorización que proyectan las respuestas en un espacio latente de alta dimensionalidad mediante el modelo `mxbai-embed-large` [32]. El código de integración de este cliente de vectorización automatizado se detalla en el Anexo H. Esta infraestructura algorítmica establece la base técnica necesaria para calcular posteriormente métricas de distancia angular (similitud del coseno) e índices de fidelidad de fuente en asistentes virtuales basados en RAG, las cuales se discutirán en el Capítulo 5.

4.8.2 Trazabilidad nativa del orquestador

La evaluación del comportamiento global del ecosistema requiere auditar la precisión de la delegación de tareas y el rendimiento concurrente. Para que las

pruebas integrales sean deterministas, se diseñó al orquestador con capacidades de inyección de métricas y trazabilidad distribuida.

Durante el procesamiento de los hilos de ejecución concurrente, el orquestador mide algorítmicamente el tiempo de respuesta de cada asistente invocado y preserva el segmento de texto exacto que le fue asignado. Posteriormente, el sistema inyecta estos metadatos en la estructura JSON de la respuesta colaborativa final, utilizando atributos estandarizados (`span` para la latencia temporal y `context` para la trazabilidad de la consulta). La implementación de esta inyección de eventos se expone en el Anexo H.

Gracias a esta arquitectura orientada a la observabilidad, los *scripts* de evaluación externos pueden interceptar el paquete OFP y extraer los identificadores de los asistentes (`speakerUri`), permitiendo contrastar de manera automatizada las decisiones de enrutamiento del LLM contra las dependencias esperadas.

4.8.3 Tolerancia a fallos y gestión de memoria VRAM

La ejecución masiva de pruebas automatizadas sobre un servidor de inferencia local genera una rápida saturación de la VRAM. Sin intervención, esta acumulación de tensores derivaría irremediablemente en fallos críticos por desbordamiento de memoria.

Para dotar al módulo de evaluación de tolerancia a fallos, se programó un subsistema de gestión activa del ciclo de vida de los modelos. Este componente realiza llamadas preventivas a la API del motor de inferencia (Ollama), inyectando el parámetro `"keep_alive": 0` para forzar la liberación inmediata y la descarga de los modelos de la aceleradora gráfica (ver Anexo H). Al ejecutar esta rutina de limpieza de forma sistemática al finalizar cada iteración de prueba, se garantiza un entorno de cómputo predecible para el siguiente ciclo. Esta estrategia asegura que el cálculo estadístico sobre la latencia de red no se vea sesgado por bloqueos térmicos o saturación temporal del hardware.

Capítulo 5

Evaluación y métricas del modelo de interoperabilidad

Este capítulo presenta la metodología de evaluación y los resultados obtenidos tras la implementación del modelo de interoperabilidad para asistentes virtuales heterogéneos educativos. El objetivo principal de esta evaluación es medir la eficacia, precisión y rendimiento del sistema propuesto bajo distintas cargas de trabajo y arquitecturas de modelos LLM.

Para lograr un análisis exhaustivo, las pruebas se dividieron en dos fases. La Fase 1 se centra en validar la precisión individual de los diez asistentes especialistas que componen el ecosistema, evaluando el desempeño de cada uno mediante preguntas específicas de su dominio. Posteriormente, la Fase 2 evalúa el rendimiento central del sistema: la capacidad del orquestador (actuando como enrutador semántico) para deconstruir consultas complejas y delegarlas correctamente. A través de métricas de ruteo, similitud semántica y latencia, se contrasta el desempeño de cuatro modelos LLM (Qwen, Mistral, Gemma y Llama) para determinar qué arquitectura presenta la mayor idoneidad para tareas de orquestación multi-asistente basadas en el protocolo OFP.

5.1 Configuración del entorno de evaluación

Para llevar a cabo la evaluación del modelo propuesto, se diseñó un entorno de pruebas centralizado basado en un orquestador, el cual actúa como enrutador principal utilizando OFP para la estandarización de los mensajes. El modelo de interoperabilidad cuenta con 10 asistentes educativos especializados. Los asistentes de Historia, Computación, Programación, Química, Español y Álgebra comparten la misma arquitectura base microservicio con *endpoint* `/preguntar` y recuperación vectorial, variando únicamente el modelo generador, el corpus documental indexado y el nivel educativo objetivo. El asistente de Traductor opera bajo dos modalidades de invocación, gestionadas por el orquestador mediante el campo `modo` del mensaje OFP: la modalidad `translator`, que procesa la traducción de un fragmento específico de la consulta, y la modalidad `translator_global`, reservada para traducir la

respuesta ensamblada completa producida por todos los asistentes que respondieron una consulta multidominio. Ambas modalidades comparten el mismo microservicio subyacente, pero difieren en el alcance del texto procesado y en su posición dentro del pipeline de ejecución: `translator_global` siempre se ejecuta como último paso, una vez que el orquestador ha recibido y concatenado las respuestas de los demás asistentes involucrados. Los asistentes de Física, Matemática —dividido en dos módulos independientes, Avanzada y Básica— y Traductor se apoyan en arquitecturas distintas, detalladas en cada caso a continuación:

1. **Física:** dirigido a nivel secundaria (grados 9° y 10°, currículo del *National Curriculum and Textbook Board*, de Bangladesh), cubre física conceptual y de resolución de problemas mediante RAG sobre el libro Physics Classes 9-10 y fuentes complementarias (Panjeree Test Paper, Teachingbd24.com), con generación a cargo de modelo de lenguaje pequeño servido vía Ollama.
2. **Historia:** cubre historia universal y contemporánea mediante RAG sobre "Historia Universal Contemporánea Quinto semestre" de la Secretaría de Educación Pública y "Toda la historia del mundo: De la prehistoria a la actualidad", con generación a cargo del modelo `gemma2:2b`.
3. **Computación:** cubre fundamentos de computación e informática (hardware y software) mediante RAG sobre "Introducción a la Informática", con generación a cargo del modelo `llama3.2:3b`.
4. **Programación:** especializado en programación y lógica computacional con énfasis en C++, mediante RAG sobre "Programación y resolución de problemas con C++", con generación a cargo del modelo `llama3.1:8b`.
5. **Matemática Avanzada:** dirigido a estudiantes desde 2.º de secundaria hasta Cálculo 1 (álgebra intermedia, trigonometría, funciones, límites, derivadas e integrales). La generación corre sobre un modelo `Ollama` configurable, respaldado por una rutina de verificación simbólica con `SymPy` para derivadas, integrales y límites. Rechaza explícitamente consultas puramente básicas, remitiéndolas a Matemática Básica.
6. **Matemática Básica:** dirigido a estudiantes desde 1º de primaria hasta 2º de secundaria (aritmética, fracciones, porcentajes, ecuaciones de primer grado, perímetros y áreas básicas). La generación corre sobre un modelo `Ollama` configurable. Rechaza explícitamente consultas avanzadas remitiéndolas a Matemática Avanzada y en consultas mixtas resuelve solo la parte básica dejando una marca de traspaso (`[BASE LISTA...]`) para que el asistente avanzado continúe.
7. **Química:** cubre elementos, átomos y reacciones mediante RAG sobre química disponible en https://cemsa.edu.mx/ y "Química general un enfoque en competencias", con generación a cargo del modelo `qwen2.5:3b`.
8. **Español:** cubre gramática, ortografía y literatura mediante RAG sobre "Nueva gramática básica de la Lengua Española" y "Ortografía de la Lengua Española", con generación a cargo del modelo `gemma2:2b`.
9. **Álgebra:** cubre álgebra y matemáticas (polinomios, factorización, ecuaciones y conceptos teóricos) mediante RAG sobre "Problemas resueltos álgebra", con generación a cargo del modelo `qwen2.5:3b`.
10. **Traducción:** asistente conversacional de aprendizaje de idiomas voz a voz,

implementado con voz-a-texto y texto-a-voz y generación mediante cualquier modelo servido vía Ollama.

Todas las iteraciones de prueba se ejecutaron utilizando el mismo orquestador y un *prompt* de sistema estandarizado. Este *prompt* instruye al LLM para actuar como enrutador, exigiendo el cumplimiento de reglas estrictas: dividir lógicamente la consulta del usuario, asignar cada segmento al asistente experto correspondiente sin omitir información y determinar el orden de ejecución (secuencial o paralelo). Para garantizar el rigor de la Fase 2, el modelo clasificador del orquestador se configuró operando bajo un parámetro de `temperature=0.0`. Esta decisión fuerza una toma de decisiones completamente determinista en la capa de inferencia, lo que permite aislar y medir exclusivamente la variabilidad estocástica en la capa de ejecución de los asistentes virtuales. Además, el orquestador maneja reglas de exclusión mutua, como la delegación de traducciones globales exclusivamente al asistente `translator_global`. Para asegurar la fiabilidad estadística y evaluar la consistencia en el seguimiento de estas instrucciones, las pruebas de la Fase 2 se ejecutaron en cinco iteraciones sobre cuatro modelos de lenguaje distintos: Gemma 9B, Llama 8B, Mistral 12B y Qwen 14B.

5.1.1 Especificaciones

Dado que una de las métricas principales de esta evaluación es la latencia global del sistema, es importante establecer las condiciones de hardware bajo las cuales se ejecutaron las pruebas, ya que el tiempo de inferencia depende directamente de los recursos computacionales disponibles. Todas las pruebas se ejecutaron en un equipo físico con las siguientes características:

- **Procesador (CPU):** Intel Core i7-9700K a 3.60 GHz
- **Memoria RAM:** 32 GB DDR4 a 3000 MHz (2 x 16 GB)
- **GPU:** NVIDIA GeForce RTX 2080 Ti con 11 GB de VRAM
- **Sistema operativo:** Fedora Linux 42 (Workstation Edition)
- **Motor de inferencia:** Ollama (versión 0.23.2), utilizado para la ejecución local de los modelos cuantizados mediante contenedores Docker.

5.1.2 Características de los modelos LLM evaluados

Para asegurar la fiabilidad estadística y evaluar la consistencia en el seguimiento de instrucciones complejas, las pruebas se ejecutaron sobre cuatro modelos LLM de arquitectura abierta distintos.

Qwen 2.5: desarrollado por Alibaba Cloud, este modelo fue entrenado mediante un extenso proceso de ajuste de instrucciones, que incluyó más de un millón de ejemplos, junto con varias etapas de aprendizaje por refuerzo. Para esta investigación resulta especialmente relevante su capacidad nativa para analizar datos estructurados y generar salidas precisas en formato JSON, una característica indispensable para cumplir de forma estricta con el protocolo OFP. Gracias a sus 14 mil millones de parámetros, el modelo logra mantener una alta fidelidad al seguir reglas de exclusión

mutua incluso ante consultas complejas, sin que su comprensión del lenguaje se vea degradada [98].

Mistral NeMo: fruto de la colaboración entre Mistral AI y NVIDIA, este modelo incorpora una ventana de contexto nativa de 128,000 unidades de texto (*tokens*) y un tokenizador propio, diseñado para procesar el lenguaje de manera más eficiente. A nivel arquitectónico emplea atención de consultas agrupadas para optimizar tanto el uso de memoria como la velocidad de inferencia. Su inclusión en este estudio permite analizar cómo una arquitectura optimizada para el razonamiento lógico y la retención de contextos extensos responde ante las instrucciones del orquestador [99].

Gemma 2: construido sobre la misma base de investigación que sustenta a la familia Gemini de Google DeepMind, Gemma 2 incorpora técnicas avanzadas como la atención de ventana deslizante y una limitación suave de los valores *logit*, pensadas para estabilizar tanto el entrenamiento como la inferencia. Destaca por ofrecer una capacidad de razonamiento desproporcionadamente alta en relación con su tamaño, lo que permite evaluar cuál es el número mínimo de parámetros necesario para realizar divisiones lógicas de instrucciones sin recurrir a arquitecturas de mayor escala [100].

Llama 3: desarrollado por Meta AI, este modelo representa el estado del arte entre los modelos de un solo dígito de miles de millones de parámetros, habiendo sido entrenado con más de 15 billones de unidades de texto (*tokens*). Emplea un vocabulario ampliado de 128,000 *tokens*, lo que le permite codificar el lenguaje de manera más eficiente y utiliza también atención de consultas agrupadas como mecanismo estándar. Su evaluación en esta fase sirve como punto de referencia mínimo para observar el equilibrio entre la mayor velocidad de inferencia y la capacidad de seguir reglas de enrutamiento complejas, incluso bajo condiciones de cuantización severa [101].

Las características fundamentales de los modelos seleccionados se detallan en la Tabla 5.1.

Tabla 5.1: Especificaciones técnicas de los modelos LLM evaluados.

Modelo LLM	**Parámetros**	**Ventana de contexto**	**Desarrollador**
Qwen 2.5	14 mil millones	32.768 *tokens*[*]	Alibaba Cloud
Mistral NeMo	12 mil millones	128.000 *tokens*	Mistral AI
Gemma 2	9 mil millones	8.192 *tokens*	Google DeepMind
Llama 3	8 mil millones	8.192 *tokens*	Meta AI

[*]Ventana nativa: mediante la técnica de extrapolación YaRN puede extenderse hasta 128.000 *tokens*.

La selección de estos modelos LLM permite analizar cómo el tamaño de la red neuronal desde 8 hasta 14 mil millones de parámetros puede influir en la capacidad de razonamiento lógico, la preservación de las reglas de exclusión mutua definidas en el *prompt* y la latencia general del orquestador.

5.2 Definición de métricas de evaluación

Tanto la validación del desempeño individual de los asistentes como la capacidad del orquestador para deconstruir la consulta y asignar los segmentos a los especialistas adecuados, se modelan formalmente utilizando principios de clasificación multietiqueta y recuperación de información [102].

Es crucial establecer que los conceptos de Verdadero Positivo, Falso Negativo y Falso Positivo, denotados por *VP, FN y FP*, respectivamente, adoptan interpretaciones operativas distintas dependiendo de la fase de evaluación, una práctica metodológica estandarizada en la evaluación de sistemas basados en LLM con uso de herramientas externas [103]:

Para la Fase 1, rendimiento individual, las métricas se aplican a nivel de respuesta aislada. Dado que el asistente invocado es siempre el correcto por diseño, no existe la categoría de *FP* (asistentes invocados incorrectamente). Para superar las limitaciones de las métricas de coincidencia léxica estricta, una respuesta se clasifica como *VP* si su similitud semántica con la referencia alcanza o supera el umbral de 0.75 y como *FN* si no logra alcanzarlo, lo cual puede deberse, en términos generales, a la omisión de contenido relevante o a una desviación semántica respecto a la referencia. Este criterio de clasificación basado en similitud de *embeddings* se apoya en metodologías consolidadas de evaluación de generación de texto mediante representaciones vectoriales [104].

Para la Fase 2, orquestación, el problema se aborda como una clasificación multietiqueta a nivel de sistema. Aquí, las métricas se derivan de la matriz de confusión de la invocación de asistentes, penalizando específicamente las alucinaciones de orquestación [105]:

- **Precisión (*Precision*):** mide la proporción de asistentes invocados por el orquestador que eran realmente necesarios para resolver la consulta. Penaliza las alucinaciones del modelo de interoperabilidad al inventar o llamar asistentes irrelevantes:
$$P = \frac{VP}{VP + FP} \tag{1}$$
donde *VP* representa a los asistentes correctamente seleccionados y *FP* a los asistentes invocados innecesariamente.
- **Exhaustividad (*Recall*):** Mide la proporción de asistentes necesarios que el orquestador logró identificar y llamar exitosamente. Penaliza la omisión de información:
$$R = \frac{VP}{VP + FN} \tag{2}$$
donde *FN* representa a los asistentes que debieron ser invocados pero fueron omitidos.
- **F1-Score:** Al existir un compromiso técnico entre la Precisión y la Exhaustividad, se utiliza la media armónica de ambos valores para obtener una métrica única y balanceada. Esta es la métrica principal para determinar qué modelo

LLM comprendió mejor las reglas de exclusión mutua y lógica del *prompt*:

$$\text{F1-Score} = 2 \cdot \frac{P \cdot R}{P + R} \tag{3}$$

5.2.1 Similitud semántica

Para evaluar si la respuesta final ensamblada por el sistema conserva la fidelidad de la información esperada, se comparó vectorialmente contra una respuesta ideal de referencia. A diferencia de las métricas de coincidencia exacta de texto, la Similitud Coseno evalúa la equivalencia semántica transformando los textos en vectores algebraicos espaciales [27]:

$$\text{Similitud}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} \tag{4}$$

donde A y B son los vectores multidimensionales de la respuesta generada y la respuesta ideal, respectivamente. Un valor cercano a 1.0 indica una preservación del significado casi perfecta, mitigando las variaciones sintácticas introducidas por los distintos modelos LLM.

5.2.2 Latencia global

El tiempo de ejecución es un factor restrictivo crítico para la viabilidad de los sistemas multi-asistente en entornos de producción. La latencia global (L_{global}) se define como el tiempo total transcurrido desde que el orquestador recibe la consulta inicial (t_{inicio}) hasta que emite la respuesta final ensamblada (t_{fin}), medida en segundos:

$$L_{global} = t_{fin} - t_{inicio} \tag{5}$$

Esta métrica engloba integralmente el tiempo de inferencia del modelo LLM, la Latencia de red, la ejecución concurrente o secuencial de los asistentes especialistas subyacentes y el procesamiento de los mensajes bajo el protocolo OFP.

5.2.3 Carga cognitiva percibida

Además de las métricas técnicas se consideró necesario incorporar una medida subjetiva del esfuerzo mental que el sistema demanda al usuario final, dado que la fragmentación entre asistentes virtuales heterogéneos genera una carga cognitiva adicional al alternar entre herramientas. Para cuantificar esta dimensión se utilizó el *NASA Task Load Index* (NASA-TLX) [106], un instrumento estandarizado desarrollado por el *Human Performance Group* de la NASA y ampliamente adoptado en la evaluación de interacción humano-computadora para medir la carga de trabajo percibida durante la ejecución de una tarea.

El NASA-TLX evalúa la carga cognitiva a través de seis subescalas, cada una calificada por el participante en una escala continua de 0 a 100 (en incrementos de cinco) inmediatamente después de interactuar con el sistema:

1. **Demanda mental**: cantidad de actividad mental y perceptual requerida (pensar, decidir, calcular, recordar, buscar, etc.).
2. **Demanda física**: cantidad de actividad física requerida.
3. **Demanda temporal**: presión percibida por el ritmo o la velocidad a la que ocurrió la tarea.
4. **Rendimiento (desempeño propio)**: qué tan exitoso considera el participante que fue al lograr los objetivos de la tarea.
5. **Esfuerzo**: cuánto tuvo que trabajar el participante (mental y físicamente) para alcanzar su nivel de desempeño.
6. **Nivel de frustración**: qué tan inseguro, irritado, estresado o molesto se sintió el participante durante la tarea, frente a qué tan seguro, satisfecho o relajado.

En su versión completa (*Weighted NASA-TLX*), el instrumento incluye una fase adicional de comparaciones pareadas entre las seis subescalas para obtener un peso relativo por participante. Para esta evaluación se optó por la variante *Raw TLX* (RTLX) [107], que omite la ponderación por pares y calcula la carga global como el promedio aritmético simple de las seis subescalas:

$$\text{RTLX} = \frac{1}{6}\sum_{i=1}^{6} S_i \tag{6}$$

donde S_i es la puntuación (0–100) obtenida en la subescala i. Esta variante se prefirió por dos razones: (1) reduce significativamente el tiempo de aplicación por participante, lo cual es relevante dado el tamaño de la muestra evaluada y (2) estudios posteriores al instrumento original muestran que el RTLX correlaciona fuertemente con la versión ponderada en tareas de complejidad moderada, sin pérdida sustancial de validez [107].

A modo de ilustración, la Tabla 5.2 muestra el cálculo aplicado a las seis puntuaciones registradas por un participante (P02, Condición A (manual), consulta H06), tal como fueron capturadas en el instrumento de recolección.

Tabla 5.2: Ejemplo de cálculo del RTLX a partir de puntuaciones registradas (P02, Condición A, consulta H06).

Subescala (S_i)	Puntuación (0–100)
Demanda mental	80
Demanda física	30
Demanda temporal	70
Rendimiento	20
Esfuerzo	90
Frustración	70
Suma $\sum_{i=1}^{6} S_i$	**360**

Aplicando la Ecuación (6):

$$\text{RTLX} = \frac{80 + 30 + 70 + 20 + 90 + 70}{6} = \frac{360}{6} = 60$$

Este mismo procedimiento se aplicó de forma idéntica a cada una de las 32 sesiones registradas (16 participantes × 2 condiciones, manual y orquestada). El instrumento aplicado, en su versión traducida al español, se incluye en el Anexo J.

5.3 Resultados de la Fase 1: rendimiento individual de los asistentes especializados

La Fase 1 tiene como objetivo establecer una línea base de desempeño para cada asistente especializado operando de manera aislada, antes de su integración al orquestador multi-asistente bajo el protocolo OFP. Para ello, cada uno de los diez asistentes especializados respondió de forma directa las 15 preguntas de su dominio contenidas en el conjunto de datos individual de validación, bajo cada uno de los cuatro modelos LLM evaluados. Cada respuesta se comparó mediante Similitud Coseno contra la respuesta de referencia extraída de la fuente bibliográfica correspondiente, clasificándose como *VP* si la similitud fue igual o superior a 0.75, o como *FN* en caso contrario.

La elección de este umbral específico de 0.75 responde a la necesidad de equilibrar el rigor de la evaluación con la flexibilidad generativa de los modelos LLM. Debido a la naturaleza estocástica de estos modelos, las respuestas legítimas y correctas rara vez coinciden palabra por palabra con la referencia, por lo que una métrica de coincidencia exacta de texto resultaría ineficaz al penalizar la paráfrasis válida. No obstante, en los espacios vectoriales de los modelos de *embeddings*, es habitual que textos que simplemente abordan el mismo tema general arrojen puntuaciones positivas bajas. Fijar un umbral excesivamente permisivo, e.g., 0.60, daría lugar a valores de *FP* al validar respuestas tangenciales, incompletas o con alucinaciones parciales. Por lo tanto, el valor de 0.75 se establece como un punto de equilibrio: es lo suficientemente flexible para permitir variaciones legítimas en la estructura gramatical, pero lo bastante estricto para garantizar que los hechos fundamentales y el núcleo semántico requeridos estén presentes en la respuesta final [108].

La Tabla 5.3 presenta los resultados de la Fase 1 de evaluación, en la cual cada uno de los diez asistentes especialistas –Historia, Física, Traductor, Química, Español, Programación, Computación, Álgebra, Matemática Básica y Matemática Avanzada– fue evaluado de manera aislada e independiente, sin intervención del componente de orquestación, sobre los cuatro modelos LLM considerados en este estudio (Qwen 14B, Mistral 12B, Gemma 9B y LLaMA 8B). Este diseño aislado tiene como propósito establecer una línea base de rendimiento intrínseco para cada asistente, libre de los errores de clasificación que introduce el orquestador multi-asistente, los cuales se abordan en la Fase 2. Para cada una de las cuarenta combinaciones asistente-modelo se reportan cinco métricas: Precisión y Exhaustividad (*Recall*), calculadas sobre una clasificación binaria de acierto/error derivada de un umbral de Similitud Coseno; el F1-Score como media armónica de las dos anteriores; la Similitud Coseno Media entre

la respuesta generada y la respuesta de referencia, como medida continua de calidad semántica independiente del umbral binario; y la Latencia promedio de respuesta, en segundos, como indicador de costo computacional. Como se observa en la tercera columna, la Precisión alcanza un valor de 1.000 en la totalidad de los escenarios evaluados, un resultado que, lejos de indicar un desempeño perfecto, constituye un artefacto del diseño metodológico de esta fase. Este resultado es consecuencia directa de la definición operativa de los conceptos de clasificación adoptada para esta fase (véase la Sección 5.2): al no existir, por diseño experimental, la categoría de los valores *FP*— toda consulta se envía directamente al asistente correspondiente, sin mediar una decisión de orquestación que pueda errar —, la Precisión, dada por la Ecuación (1), se reduce algebraicamente a $VP/\,VP = 1$ en cada uno de los escenarios evaluados. Este valor constante no debe interpretarse, por tanto, como evidencia de un desempeño óptimo del sistema, sino como un artefacto aritmético derivado de la ausencia estructural del denominador que permitiría diferenciar el desempeño entre asistentes o entre modelos LLM.

Dicha ausencia de variabilidad no se traslada, sin embargo, a la Exhaustividad ni a la Similitud Coseno, dado que ambas métricas dependen de los valores *FN* — i.e., de los casos en que la similitud semántica de la respuesta no alcanza el umbral de 0.75 definido previamente — y estos sí ocurren con frecuencia variable según la calidad del *corpus* documental y el modelo generador de cada asistente. Por ello, ambas métricas logran capturar diferencias reales de desempeño entre los diez asistentes evaluados, tal como se observa en las Figuras 5.1 y 5.2.

Tabla 5.3: Rendimiento base de asistentes especialistas en los modelos LLM evaluados.

Asistente	**Modelo LLM**	**Precisión**	**Exhaustividad**	**F1-Score**	**Sim. Coseno**	**Latencia (s)**
Historia	Qwen 14B	1.000	0.333	0.500	0.691	24.90
	Mistral 12B	1.000	0.467	0.636	0.720	24.04
	Gemma 9B	1.000	0.400	0.571	0.705	25.02
	LLaMA 8B	1.000	0.333	0.500	0.713	26.12
Física	Qwen 14B	1.000	0.667	0.800	0.784	35.37
	Mistral 12B	1.000	0.667	0.800	0.770	35.80
	Gemma 9B	1.000	0.800	0.889	0.788	35.27
	LLaMA 8B	1.000	0.733	0.846	0.785	34.51
Traductor	Qwen 14B	1.000	0.800	0.889	0.800	22.09
	Mistral 12B	1.000	0.733	0.846	0.802	20.40
	Gemma 9B	1.000	0.733	0.846	0.791	19.13
	LLaMA 8B	1.000	0.667	0.800	0.780	22.25
Química	Qwen 14B	1.000	0.733	0.846	0.785	33.41
	Mistral 12B	1.000	0.800	0.889	0.808	34.32
	Gemma 9B	1.000	0.800	0.889	0.805	35.00
	LLaMA 8B	1.000	0.667	0.800	0.781	38.16
Español	Qwen 14B	1.000	0.846	0.917	0.681	27.10
	Mistral 12B	1.000	0.846	0.917	0.686	27.62
	Gemma 9B	1.000	0.692	0.818	0.665	25.38
	LLaMA 8B	1.000	0.923	0.960	0.696	27.05
Programación	Qwen 14B	1.000	0.933	0.966	0.823	110.39
	Mistral 12B	1.000	0.867	0.929	0.820	99.88
	Gemma 9B	1.000	0.867	0.929	0.816	87.14
	LLaMA 8B	1.000	0.867	0.929	0.825	94.63
Computación	Qwen 14B	1.000	1.000	1.000	0.865	38.07
	Mistral 12B	1.000	1.000	1.000	0.852	33.16
	Gemma 9B	1.000	1.000	1.000	0.867	37.25
	LLaMA 8B	1.000	1.000	1.000	0.857	36.84
Álgebra	Qwen 14B	1.000	0.857	0.923	0.759	44.66
	Mistral 12B	1.000	0.857	0.923	0.767	42.24
	Gemma 9B	1.000	0.857	0.923	0.781	43.32
	LLaMA 8B	1.000	0.714	0.833	0.760	42.27
Matemática Básica	Qwen 14B	1.000	0.933	0.966	0.500	23.20
	Mistral 12B	1.000	0.800	0.889	0.503	21.40
	Gemma 9B	1.000	0.867	0.929	0.490	24.10
	LLaMA 8B	1.000	0.867	0.929	0.494	23.41
Matemática Avanzada	Qwen 14B	1.000	0.800	0.889	0.594	54.08
	Mistral 12B	1.000	0.733	0.846	0.619	50.81
	Gemma 9B	1.000	1.000	1.000	0.604	51.44
	LLaMA 8B	1.000	0.933	0.966	0.614	51.57

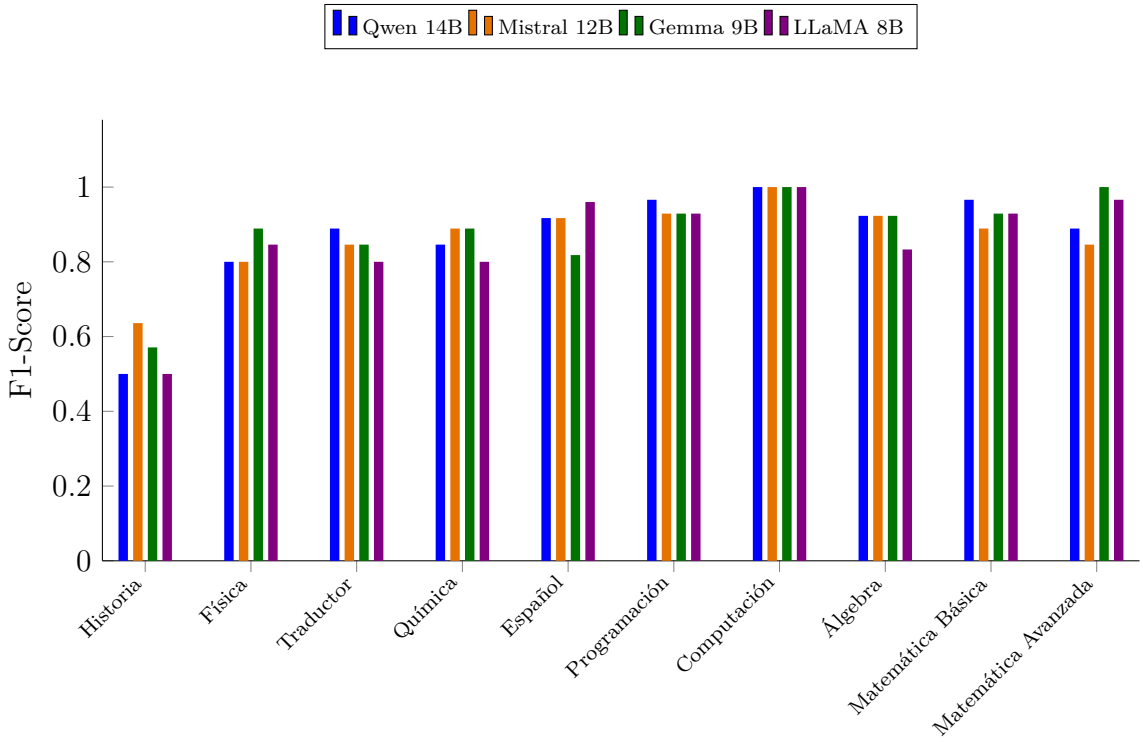


Figura 5.1: F1-Score por asistente especialista y modelo LLM.

La Figura 5.1 muestra que el F1-Score varía fuertemente entre asistentes, no entre modelos LLM: para cada asistente, los cuatro modelos LLM producen valores casi idénticos, mientras que entre asistentes la diferencia llega a 0.5 puntos. Historia obtiene el F1-Score más bajo de los diez asistentes en los cuatro modelos LLM (0.500–0.636); una hipótesis plausible es que su *corpus*, de naturaleza narrativa y extensa, admite múltiples formulaciones semánticamente válidas de una misma respuesta, lo que dificulta alcanzar el umbral de similitud fijo frente a una única referencia bibliográfica. Computación, en el extremo opuesto, alcanza F1-Score=1.000 en los cuatro modelos LLM, lo cual es consistente con la naturaleza más determinística de sus preguntas: las definiciones técnicas de hardware y software tienden a tener una única formulación canónica, lo que reduce el espacio de variación léxica frente a la referencia y facilita superar el umbral de similitud. Llama la atención que Matemática Básica (0.889–0.966) y Matemática Avanzada (0.846–1.000) se ubican también entre los asistentes de mejor desempeño según esta métrica, un resultado que, como se verá en la Figura 5.4, requiere una interpretación cuidadosa.

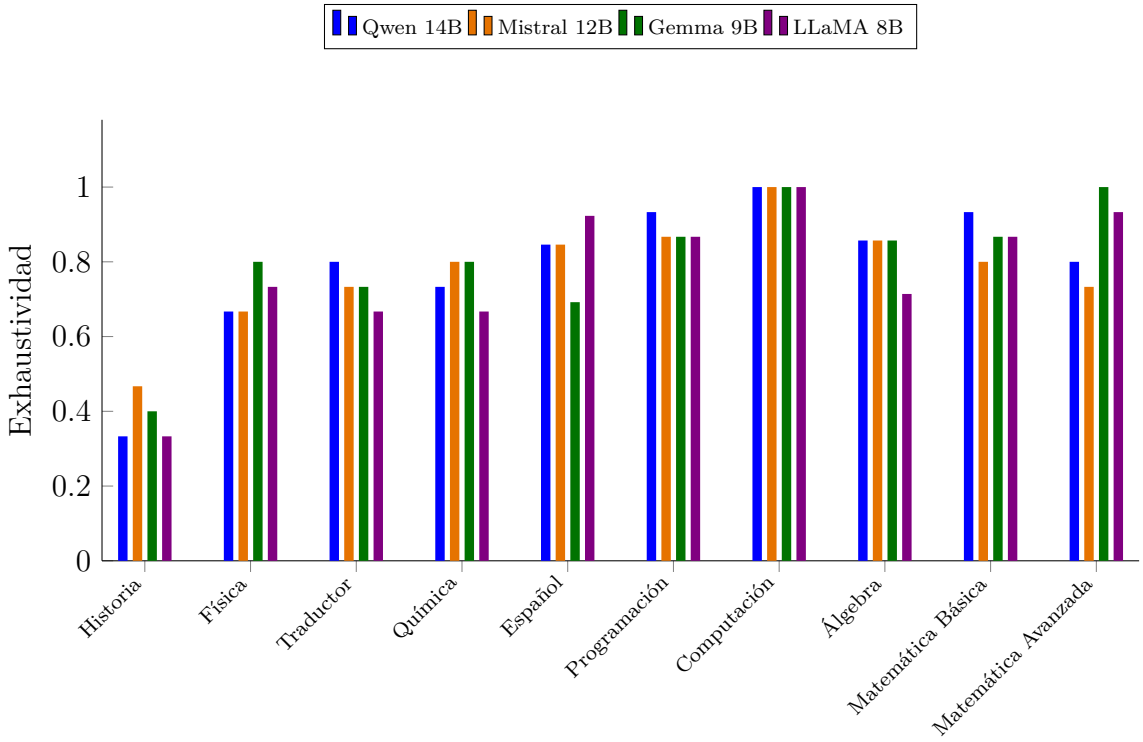


Figura 5.2: Exhaustividad por asistente especialista y modelo LLM.

El patrón de Exhaustividad en la Figura 5.2 revela una estructura de tres niveles claramente diferenciada entre los diez asistentes especialistas. En el extremo inferior, Historia exhibe el Exhaustividad más baja del conjunto (0.333–0.467) de manera uniforme en los cuatro modelos LLM, lo que indica un déficit sistemático de cobertura semántica independiente de la arquitectura subyacente. En el extremo opuesto, Computación alcanza Exhaustividad=1.000 en los cuatro modelos sin excepción, constituyendo el único asistente con desempeño perfecto y homogéneo. Entre ambos extremos se distingue un grupo intermedio –Física, Traductor, Química y Álgebra– cuyos valores oscilan en una banda relativamente estrecha (0.667–0.857) y un grupo superior –Programación y Matemática Básica– que se mantiene consistentemente por encima de 0.800.

Resulta particularmente relevante la dispersión inter-modelo observada en Español (0.692–0.923) y Matemática Avanzada (0.733–1.000), pues contrasta con la notable estabilidad que caracteriza al resto de los asistentes. En ambos casos, Gemma 9B se aparta del comportamiento de los demás modelos LLM: obtiene la Exhaustividad más baja en Español (0.692) pero más alta en Matemática Avanzada (1.000), lo que sugiere una sensibilidad específica de este modelo LLM a la naturaleza del dominio evaluado, más que una tendencia general de subdesempeño o sobredesempeño.

Esta jerarquía entre asistentes reproduce exactamente el ordenamiento observado en el F1-Score (Figura 5.1), lo cual no es una coincidencia empírica sino una consecuencia algebraica directa: dado que la Precisión es igual a 1.000 en todos los escenarios de la Fase 1 (Tabla 5.3), el F1-Score queda totalmente determinado por la Exhaustividad mediante la relación $F1 - Score = 2R/(1 + R)$. En consecuencia,

la Figura 5.2 no aporta información adicional sobre el ordenamiento relativo de los asistentes respecto a la Figura 5.1, pero sí permite identificar en qué medida cada modelo LLM individual se desvía del comportamiento esperado para un asistente dado –como en los casos de Español y Matemática Avanzada– aspecto que la métrica F1-Score comprimida tiende a oscurecer.

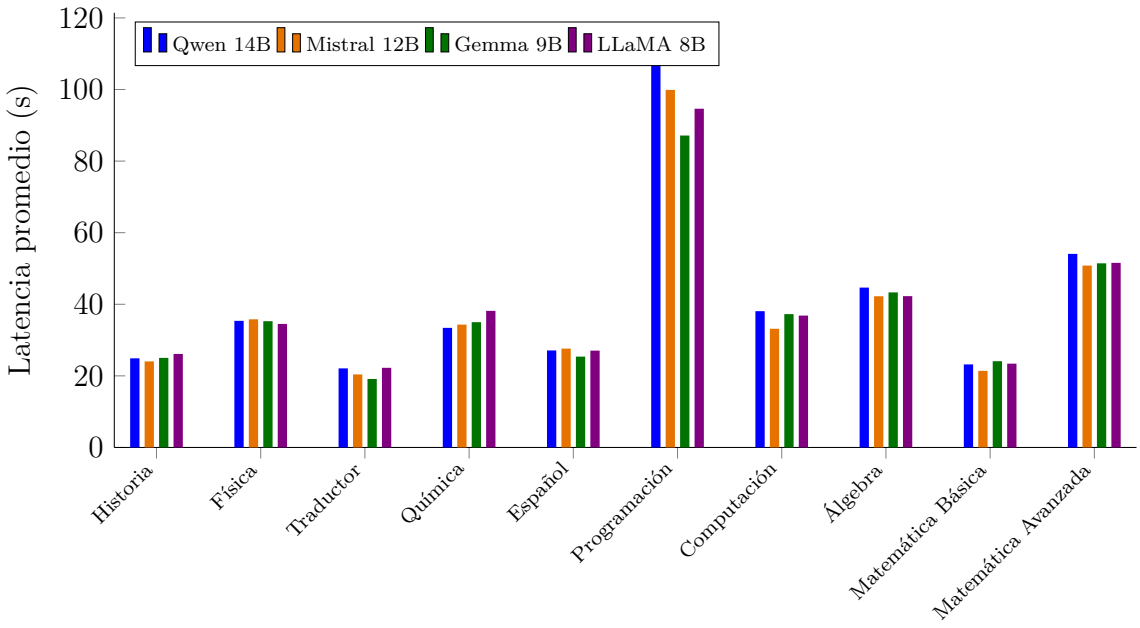


Figura 5.3: Latencia promedio por asistente especialista y modelo LLM.

La Figura 5.3 revela una estructura jerárquica de costo computacional claramente diferenciada entre los asistentes especialistas. Un primer grupo, compuesto por Historia, Traductor, Español y Matemática Básica, exhibe las Latencias más bajas del conjunto (19–27 s), consistentes con tareas de generación de respuesta relativamente breve. Un segundo grupo intermedio –Física, Química, Computación y Álgebra– se ubica en un rango de 33–45 s, mientras que Matemática Avanzada constituye un caso límite superior dentro del comportamiento normal del sistema, con valores de 51–54 s. Frente a esta progresión gradual, Programación se distingue como una anomalía manifiesta: con Latencias de 87–110 s, supera en más del doble al asistente inmediatamente más lento (Matemática Avanzada) y en hasta seis veces a los asistentes más rápidos, configurando un sobrecosto que no guarda proporción con el resto de la escala observada.

Un segundo hallazgo relevante de la Figura 5.3 concierne a la sensibilidad de la Latencia frente a la elección del modelo LLM. En nueve de los diez asistentes, la Latencia se mantiene prácticamente invariante entre modelos LLM, con diferencias inter-modelo que no superan los 5 segundos en términos absolutos (e.g., 1.29 s en Física o 4.91 s en Computación). Programación rompe nuevamente este patrón: la diferencia entre su configuración más lenta (Qwen 14B, 110.39 s) y la más rápida (Gemma 9B, 87.14 s) asciende a 23.25 s, equivalente a un 27 % de variación relativa, la mayor registrada en todo el estudio tanto en términos absolutos como proporcionales. Este comportamiento es consistente con la hipótesis de que la naturaleza de la tarea

asignada a este asistente —la generación de bloques de código sintácticamente válidos— impone una longitud de salida considerablemente mayor que la prosa explicativa producida por los demás asistentes: cada respuesta incluye la declaración de variables, la lógica del algoritmo, comentarios y, con frecuencia, un fragmento de ejecución o verificación, lo que multiplica el número de *tokens* generados y amplifica cualquier diferencia de velocidad de inferencia entre arquitecturas. Esta sensibilidad diferencial de Programación al modelo LLM subyacente, frente a la invarianza observada en los nueve asistentes restantes, se retoma en la discusión sobre el balance entre precisión semántica y eficiencia computacional.

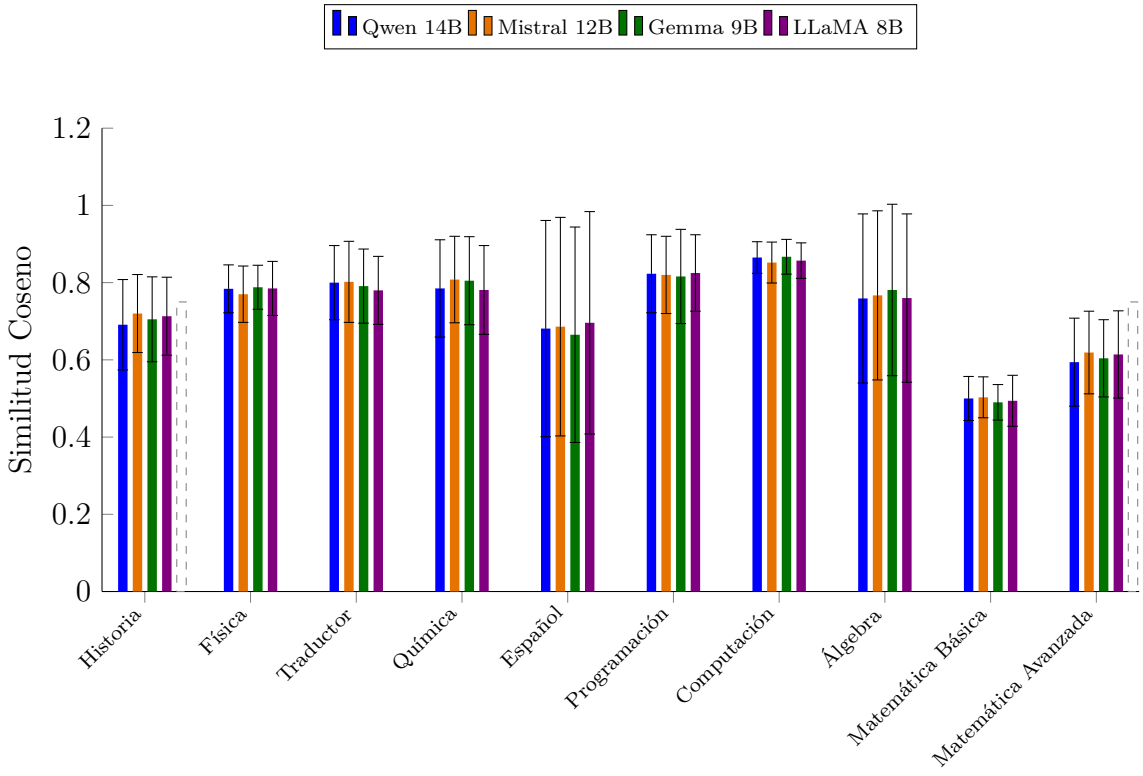


Figura 5.4: Similitud Coseno Media por asistente y modelo LLM, con barras de error de ±1 desviación estándar. La línea discontinua marca el umbral de clasificación VP/FN (0.75) (ver en Sección 5.3)

La Similitud Coseno Media (Figura 5.4) permite matizar varios de los hallazgos anteriores. Historia, el asistente con menor F1-Score (Figura 5.1), presenta sin embargo una Similitud Coseno Media de 0.691–0.720, apenas por debajo del umbral de clasificación VP/FN (0.75), lo que indica que sus respuestas son semánticamente cercanas a la referencia y que el corte binario las penaliza como fallo total a pesar de su proximidad semántica. Las implicaciones metodológicas de este hallazgo se analizan en la Sección 5.9.

Español y Álgebra muestran, en los cuatro modelos LLM, la desviación estándar más alta de Similitud Coseno de los diez asistentes (0.218–0.288), frente a un rango de 0.041–0.126 en el resto de los asistentes evaluados. La consistencia de esta brecha entre modelos sugiere que la causa es la heterogeneidad del propio conjunto de preguntas

o de la fuente documental de Español y Álgebra (algunas respuestas muy precisas, otras muy alejadas), y no una limitación específica de algún modelo LLM.

Computación, además de su F1-Score=1.000 (Figura 5.1), presenta aquí Similitud Coseno consistentemente alta (0.852–0.867) y la menor dispersión de todos los asistentes (desviación estándar de 0.041 a 0.053). Esta coincidencia entre F1-Score máximo, similitud alta y dispersión mínima representa el techo de desempeño observado en esta fase aislada y es consistente con la idea de que la calidad del asistente especialista –su fuente documental y el diseño de su *prompt*– constituye un factor con peso propio, independiente del modelo de interoperabilidad que lo invoque.

El hallazgo más relevante de la Figura 5.4 es la inversión de calibración que exhiben los asistentes de contenido matemático: Matemática Básica y Matemática Avanzada obtienen la Similitud Coseno Media más baja del estudio (0.490–0.503 y 0.594–0.619, respectivamente) –muy por debajo del umbral de 0.75– a pesar de tener, según la Figura 5.1, dos de los F1-Score más altos (0.889–0.966 y 0.846–1.000). Esta disociación entre ambas métricas opera en sentido contrario a la observada en Historia –donde un F1-Score bajo coexistía con una similitud apenas por debajo del umbral– y lo hace con una brecha considerablemente mayor, dado que aquí un F1-Score entre los más altos del estudio coexiste con una similitud muy alejada del umbral de clasificación. La explicación de esta aparente contradicción entre F1-Score alto y Similitud Coseno baja se desarrolla en la Sección 5.9.

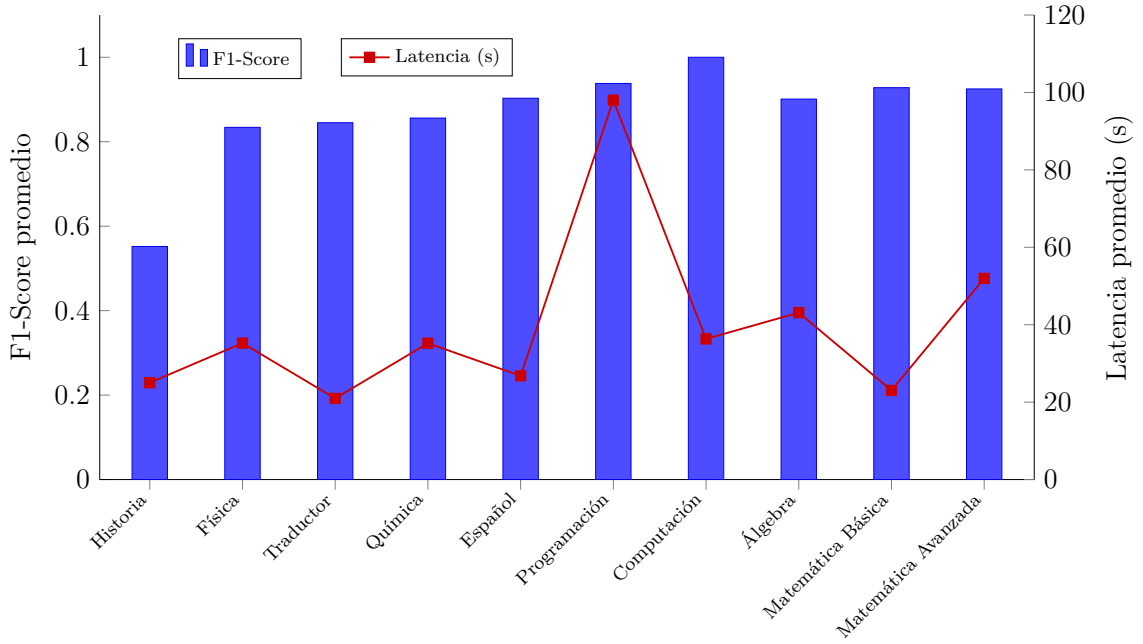


Figura 5.5: Comparación entre el F1-Score promedio y la Latencia promedio por asistente especialista (ejes Y independientes).

La Figura 5.5 sintetiza los dos ejes de desempeño discutidos anteriormente mediante un gráfico de doble eje, lo que permite verificar visualmente si el F1-Score guarda alguna relación con el costo computacional Latencia de cada asistente. El panorama general es de independencia casi total entre ambas dimensiones: asistentes con

Latencias muy distintas alcanzan F1-Score igualmente altos. Computación (Latencia promedio de 36 s) y Matemática Avanzada (52 s, casi el doble) logran ambos un F1-Score cercano o igual a 1.000; de manera similar, Matemática Básica (23 s, una de las más bajas del conjunto) y Álgebra (43 s) obtienen F1-Score igualmente altos (0.928 y 0.901 en promedio, respectivamente) a pesar de que su Latencia difiere sustancialmente. Esta disociación resulta particularmente notable en Matemática Básica y Matemática Avanzada, cuyo F1-Score elevado contrasta, como se discutió en la Figura 5.4, con la Similitud Coseno Media más baja del estudio: ni la Latencia ni la Similitud Coseno parecen ser predictores confiables del comportamiento real de Precisión y Exhaustividad de estos dos asistentes.

Historia es el único asistente que combina una Latencia baja (25 s, una de las menores del conjunto) con el F1-Score más bajo del estudio (0.552 en promedio), lo que lo distingue como un caso real de bajo desempeño –o, como se discutió previamente, de desempeño mal capturado por el umbral de clasificación (Figura 5.4)– y no de una tarea intrínsecamente costosa que el sistema resuelve mal por falta de tiempo de procesamiento.

En el extremo opuesto, el pico de la línea roja demuestra que Programación se separa claramente del resto en el eje de Latencia (con un promedio de 98 s, más del doble que el siguiente asistente más lento) sin que ello comprometa el nivel de su barra de F1-Score (0.938), lo cual sugiere que su costo computacional no proviene de una mayor dificultad para acertar, sino del tiempo de procesamiento que demanda la tarea, posiblemente ligado a la generación o verificación de código.

Con los cuatro modelos LLM disponibles, es posible examinar por primera vez si el desempeño aislado de los asistentes especializados guarda relación con el número de parámetros del modelo LLM que los respalda. La Tabla 5.4 resume los promedios globales (sobre los diez asistentes) para cada modelo LLM.

Tabla 5.4: Promedios globales por modelo LLM, Fase 1 (síntesis de la Tabla 5.3).

Modelo LLM	**Exhaustividad Media**	**F1-Score Medio**	**Sim. Coseno Media**	**Latencia Media (s)**
Qwen 14B	0.790	0.870	0.728	41.33
Mistral 12B	0.777	0.868	0.735	38.97
Gemma 9B	0.802	0.879	0.731	38.30
LLaMA 8B	0.770	0.856	0.730	39.68

Como se observa en la Tabla 5.4, los promedios globales de la Fase 1 muestran una relación monótona entre el tamaño del modelo LLM y el desempeño: el F1-Score medio más alto corresponde a Gemma 9B (0.879), a pesar de no ser ni el modelo más grande (Qwen 14B, 0.870) ni el más pequeño (LLaMA 8B, 0.856) evaluado; Qwen 14B y Mistral 12B obtienen valores intermedios prácticamente empatados (0.870 y 0.868, respectivamente) pese a diferir en 2 000 millones de parámetros; y LLaMA 8B –el modelo más pequeño– obtiene el promedio más bajo (0.856), aunque la diferencia frente al más alto es de apenas 0.023 puntos de F1-Score. Un patrón equivalente se observa en la Exhaustividad media, donde Gemma 9B vuelve a liderar (0.802), seguido de Qwen 14B (0.790), Mistral 12B (0.777) y LLaMA 8B (0.770). En Latencia, el modelo LLM más rápido en promedio es, de nuevo, Gemma 9B (38.30 s) y no el

modelo de menor tamaño (LLaMA 8B, 39.68 s). Bajo las condiciones evaluadas en este estudio, Gemma 9B exhibe el mejor desempeño promedio y la menor Latencia simultáneamente, sin ser el de mayor ni el de menor tamaño evaluado.

En conjunto, los resultados de la Fase 1 muestran que, bajo evaluación aislada, los diez asistentes especializados alcanzan un desempeño alto y consistente independientemente del modelo LLM utilizado, con diferencias entre modelos LLM de a lo sumo 0.023 puntos de F1-Score medio. La calidad del corpus documental y el diseño del asistente especialista –evidenciada por la brecha de hasta 0.5 puntos de F1-Score entre Historia y Computación– resulta así un factor considerablemente más determinante para el desempeño individual que el tamaño del modelo LLM dentro del rango de 8 a 14 mil millones de parámetros evaluados.

5.4 Resultados de la Fase 2: evaluación con Llama 3.1 8B

En esta subsección se presentan los resultados obtenidos durante la Fase 2 de evaluación del modelo de interoperabilidad, en la cual el orquestador empleó el modelo Llama 3.1 8B como motor para la clasificación y segmentación de la pregunta. La evaluación se ejecutó sobre el conjunto híbrido de 35 preguntas (ver Anexo I), repitiendo el proceso completo en cinco iteraciones independientes con el fin de capturar la variabilidad inherente a la generación estocástica de los modelos LLM y obtener medias y desviaciones estándar representativas de cada métrica.

5.4.1 Desempeño global

La Tabla 5.5 resume las métricas globales y de calidad semántica, consolidadas como la media y la desviación estándar de los valores obtenidos en las cinco iteraciones.

Tabla 5.5: Métricas globales del orquestador — Llama 3.1 8B (media y desviación estándar entre cinco iteraciones)

Métrica	**Media**	**Desv. estándar**
Precisión	0.4954	0.0000
Exhaustividad	0.4943	0.0000
F1-Score	0.4892	0.0000
Similitud Coseno	0.5549	0.0259
Latencia Global (segundos)	277.30	7.01

Como se observa en la Tabla 5.5, el F1-Score promedio fue de 0.4892, un valor que indica un desempeño de clasificación moderado-bajo: en promedio, algo menos de la mitad de las asignaciones asistente–consulta coinciden exactamente con lo esperado. Un dato relevante es que la Precisión, la Exhaustividad y el F1-Score presentan desviación estándar igual a cero entre las cinco iteraciones. Esto es consecuencia de la

configuración del entorno (`temperature=0.0`), la cual produce un comportamiento determinista en el clasificador: ante la misma consulta, siempre asigna el mismo conjunto de asistentes.

5.4.2 Sesgo del *prompt* de clasificación

Al analizar las 35 preguntas se encontró que en 21 de ellas (60 %) el conjunto de asistentes realmente invocado coincide de forma exacta con este ejemplo, independientemente del dominio real de la consulta (Química, Programación, Traducción, Álgebra, etc.).

Este comportamiento es consistente con el fenómeno conocido en la literatura como sesgo de anclaje al ejemplo: bajo este escenario determinista, el modelo de clasificación tiende a reproducir la estructura del ejemplo proporcionado en el *prompt* en lugar de razonar de forma independiente sobre el contenido semántico de cada consulta. La consecuencia directa de este sesgo es que el asistente `mathbuddy_avanzado` —que no aparece en el ejemplo del *prompt*—nunca fue seleccionado en ninguna de las cinco iteraciones, ni siquiera en las preguntas donde era explícitamente el asistente esperado (ver Tabla 5.6, fila Matemática Avanzada).

5.4.3 Desempeño por asistente especializado

La Tabla 5.6 presenta, para cada asistente especializado, el F1-Score y la media ± desviación estándar de la Similitud Coseno y de la Latencia, calculadas sobre las cinco iteraciones.

Tabla 5.6: Métricas por asistente especializado — Llama 3.1 8B (ordenado por F1-Score)

Asistente	**F1-Score**	**Sim. Coseno**		**Latencia (*s*)**	
		Media	Desv.	Media	Desv.
Historia	0.6667	0.5955	0.0046	40.06	0.96
Física	0.5957	0.6675	0.0067	69.03	1.31
Álgebra	0.5714	0.5325	0.0121	68.35	11.69
Matemática Básica	0.5333	0.6033	0.0140	36.46	0.92
Computación	0.4706	0.6544	0.1022	66.41	10.76
Programación	0.4706	0.5889	0.2454	135.52	17.97
Español	0.4286	0.5353	0.1446	51.54	9.21
Química	0.4000	0.4783	0.0059	40.16	4.95
Traductor	0.3333	0.6628	0.0156	7.33	0.12
Traductor Global	0.2105	0.3433	0.1919	74.68	9.01
Matemática Avanzada	0.0000	0.0000	0.0000	0.00	0.00

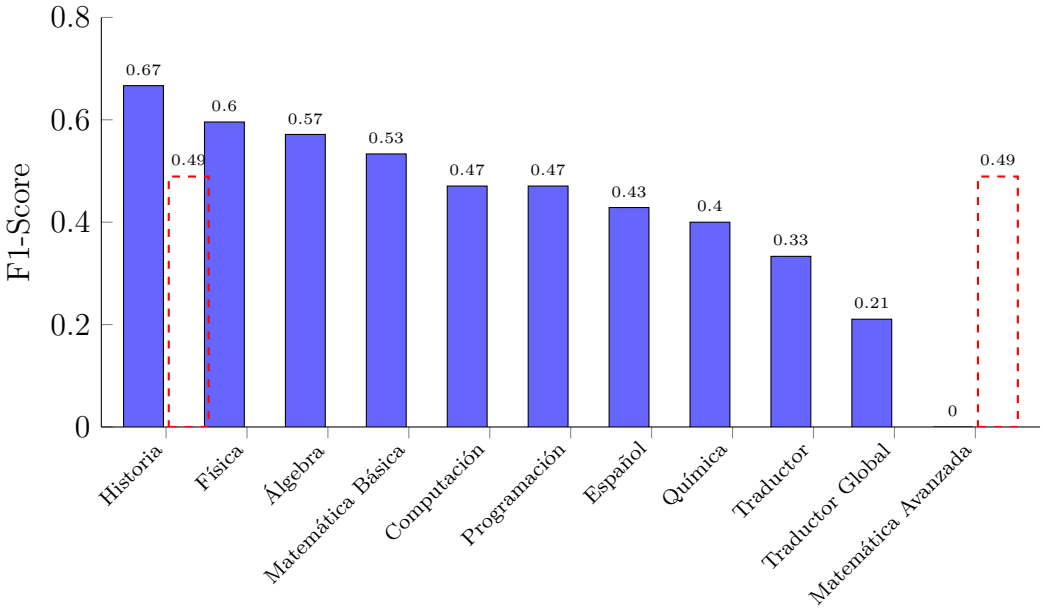


Figura 5.6: F1-Score por asistente especializado — Llama 3.1 8B. La línea punteada roja representa la media global (0.4892).

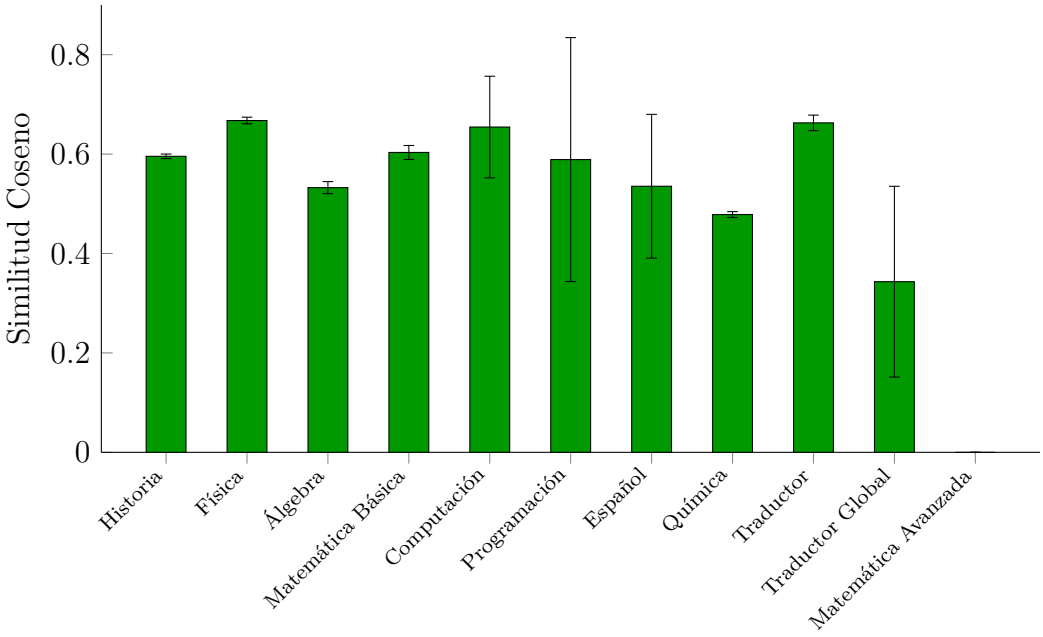


Figura 5.7: Similitud Coseno promedio por asistente especializado — Llama 3.1 8B (media ± desviación estándar entre cinco iteraciones).

Se observa que el asistente Historia obtuvo el mejor desempeño (F1-Score = 0.6667), seguido de Física (F1-Score = 0.5957) y Álgebra (F1-Score = 0.5714). En el extremo opuesto, Traductor Global (F1-Score = 0.2105) y Matemática Avanzada (F1-Score = 0.0000) presentan el desempeño más bajo, consistente con el sesgo de anclaje descrito en la sección 5.4.2.

En cuanto a la calidad semántica (Figura 5.7), los asistentes Traductor (0.6628),

Computación (0.6544) y Física (0.6675) presentan Similitudes Coseno más altas y, además, las desviaciones estándar más bajas, lo que indica respuestas tanto correctas como consistentes entre iteraciones. En contraste, Programación (0.5889) y Traductor Global (0.3433) combinan una Similitud Coseno moderada-baja con la mayor variabilidad, 0.2454 y 0.1919, respectivamente, lo que sugiere que la calidad de sus respuestas es inconsistente de una ejecución a otra, probablemente por la longitud y complejidad de las traducciones globales o de los fragmentos de código evaluados.

Respecto a la Latencia, Traductor es, por amplio margen, el asistente más rápido (7.33 s), mientras que Programación es el más lento (135.52 s) y también el que exhibe mayor variabilidad de 17.97 s, lo cual es consistente con el mecanismo de reintento ante error de dominio implementado en el orquestador.

5.4.4 Análisis de fallos persistentes

Se identificaron como fallos aquellas respuestas cuya Similitud Coseno fue igual a 0.0, ya sea por la detección de un mensaje de error interno o por una respuesta de evasión e.g., "lo siento". El número de fallos osciló entre cinco y ocho preguntas por iteración (de un total de 35) y cuatro preguntas fallaron en las cinco iteraciones sin excepción: H06, H09, H13 y H22. La Tabla 5.7 detalla estos casos.

Tabla 5.7: Preguntas con fallo persistente — Llama 3.1 8B

ID	**Pregunta**	**Asistentes esperados**	**Asistentes invocados**
H06	¿Cuál es la diferencia entre Aritmética y Álgebra, para qué sirven los operadores `new` y `delete` en C++, cuánto es 20/4, y traduce todo el resultado al portugués?	Álgebra, Programación, Matemática Básica, Traductor Global	Historia, Física, Matemática Básica, Programación, Computación, Álgebra, Traductor Global
H09	Explica qué es un átomo y sus partículas, resuelve la ecuación $x + 5 = 12$, indica cuándo llevan tilde "quién" y "cuál", e integra la función $\sin(x)$.	Química, Matemática Básica, Español, Matemática Avanzada	Matemática Básica, Física, Español, Química

Tabla 5.7: Preguntas con fallo persistente — Llama 3.1 8B (continuación).

ID	Pregunta	Asistentes esperados	Asistentes invocados
H13	¿En qué consiste el factor común en factorización, qué son los isótopos, cuál es la diferencia entre hardware y software, y cuál es la diferencia entre palabras agudas, graves y esdrújulas?	Álgebra, Química, Computación, Español	Historia, Física, Matemática Básica, Álgebra, Español, Computación
H22	¿Por qué los imperios precolombinos fueron derrotados, qué plantea la Ley de Conservación de la Energía y cuál es la diferencia entre paso por valor y por referencia en programación?	Historia, Física, Programación	Historia, Física, Matemática Básica

El análisis de estos cuatro casos revela un patrón común: las cuatro preguntas requieren la coordinación de entre tres y cuatro asistentes distintos, lo que las ubica entre las consultas de mayor complejidad de orquestación del conjunto evaluado. Adicionalmente, en tres de los cuatro casos (H06, H13 y H22) el conjunto de asistentes invocados incluye a Historia, Física y/o Matemática Básica en lugar de los asistentes verdaderamente pertinentes: Álgebra y Química en H13, o Programación en H22. Esto sugiere que el fallo de calidad semántica observado no es aleatorio, sino una consecuencia directa de un error de ruteo previo: al no invocarse el asistente correcto, la respuesta combinada se aleja semánticamente de la respuesta ideal, y el sistema termina clasificándola como evasión o error.

El caso de H09 es distinto: el conjunto de asistentes invocados (Matemática Básica, Física, Español, Química) es parcialmente correcto, pero omite a Matemática Avanzada —requerido para resolver la integral de $\sin(x)$— y en su lugar incorpora a Física, un asistente no solicitado. Esto es consistente con la hipótesis de que la incapacidad del clasificador para seleccionar Matemática Avanzada constituye, en este caso, una causa suficiente de fallo total en preguntas que dependen de ese asistente.

5.5 Resultados de la Fase 2: evaluación con Gemma 2 9B

En esta subsección se evalúa el desempeño del orquestador empleando Gemma 2 9B como motor de clasificación. Construido sobre la arquitectura Gemini, este modelo fue sometido a la misma batería de pruebas: 35 preguntas (ver Anexo I) de naturaleza híbrida iteradas cinco veces bajo la configuración determinista establecida. El objetivo principal es analizar si sus técnicas de compresión y atención de ventana deslizante le permiten manejar el ruteo lógico y la exclusión mutua de manera más eficiente que Llama 8B, su competidor directo en tamaño.

5.5.1 Desempeño global

La Tabla 5.8 resume las métricas globales y de calidad semántica, consolidadas como la media y la desviación estándar de los valores obtenidos en las cinco iteraciones.

Tabla 5.8: Métricas globales del orquestador — Gemma 2 9B (media y desviación estándar entre cinco iteraciones)

Métrica	**Media**	**Desv. estándar**
Precisión	0.8514	0.0000
Exhaustividad	0.8857	0.0000
F1-Score	0.8628	0.0000
Similitud Coseno	0.5522	0.0117
Latencia Global (segundos)	436.45	21.50

Los resultados sugieren que, dentro de este conjunto experimental, Gemma 2 9B exhibe una capacidad de razonamiento lógico notablemente alta en relación con su tamaño. Alcanzó un F1-Score de 0.8628, superando ampliamente a Llama 3.1 8B (0.4892).

Al igual que en los modelos anteriores, la desviación estándar de las métricas es exactamente cero, reafirmando el determinismo de la etapa de clasificación. Sin embargo, la Similitud Coseno se mantiene en un techo de 0.5522 ± 0.0117. Esto es consistente con la hipótesis de que el cuello de botella no radica en la selección de asistentes —que Gemma realiza con alta fidelidad —sino en el ensamblaje de la respuesta final y las capacidades intrínsecas de los asistentes subyacentes. La Latencia Global de 436.45 segundos confirma que, aunque es un modelo eficiente en enrutamiento, la orquestación paralela pesada sigue incurriendo en penalizaciones de tiempo significativas.

5.5.2 Sesgo del *prompt* de clasificación

A diferencia de Llama 3.1 8B, Gemma 2 9B no reproduce el conjunto de asistentes del ejemplo base del *prompt* de forma sistemática: el análisis de las 35 preguntas no

detectó ningún caso de coincidencia exacta con el trío Física, Historia, Matemática Básica como único conjunto seleccionado, lo que indica que el modelo es capaz de razonar de forma independiente sobre el contenido semántico de cada consulta sin quedar anclado a la estructura del ejemplo del *prompt*. Esta capacidad queda confirmada por la puntuación perfecta de ruteo obtenida en Matemática Avanzada (F1 = 1.0000, Tabla 5.9): un asistente ausente del ejemplo de referencia que Llama 8B nunca seleccionó en ninguna de sus iteraciones, pero que Gemma identifica y convoca correctamente en todas las consultas que lo requieren.

Sin embargo, la superación del sesgo de anclaje no elimina por completo los errores de clasificación: Gemma exhibe un patrón distinto de fallo, consistente en la sobre-invocación de asistentes. En lugar de reproducir ciegamente el ejemplo, el modelo tiende a incluir asistentes adicionales no solicitados, ampliando innecesariamente el conjunto de especialistas convocados. En cuanto a las reglas de exclusión mutua, Gemma demostró una comprensión casi perfecta: diferenció correctamente cuándo delegar al asistente de Traducción, (F1-Score = 1.0000) y cuándo hacerlo al Traductor Global, (F1-Score = 0.9714), lo que contrasta favorablemente con el comportamiento de Llama, que confundió o ignoró esta distinción de forma sistemática.

5.5.3 Desempeño por asistente especializado

El desglose del desempeño por asistente (Tabla 5.9 y Figuras 5.8 y 5.9) revela cómo Gemma 2 9B maneja dominios específicos y la exclusión mutua.

Tabla 5.9: Métricas por asistente especializado — Gemma 2 9B (ordenado por F1-Score)

Asistente	**F1-Score**	**Sim. Coseno**		**Latencia (*s*)**	
		Media	Desv.	Media	Desv.
Matemática Avanzada	1.0000	0.7074	0.0010	73.17	5.23
Traductor	1.0000	0.5441	0.0158	17.32	1.13
Traductor Global	0.9714	0.6035	0.0332	102.29	3.87
Programación	0.9630	0.5860	0.0680	119.40	1.46
Química	0.9091	0.6621	0.0269	39.45	2.52
Computación	0.9000	0.6267	0.0681	46.89	5.13
Historia	0.8462	0.5769	0.0229	34.99	7.00
Física	0.8333	0.6931	0.0079	36.48	1.32
Matemática Básica	0.7742	0.6542	0.0078	34.21	2.15
Álgebra	0.7059	0.3423	0.0353	89.93	2.62
Español	0.6250	0.5846	0.0222	34.93	1.21

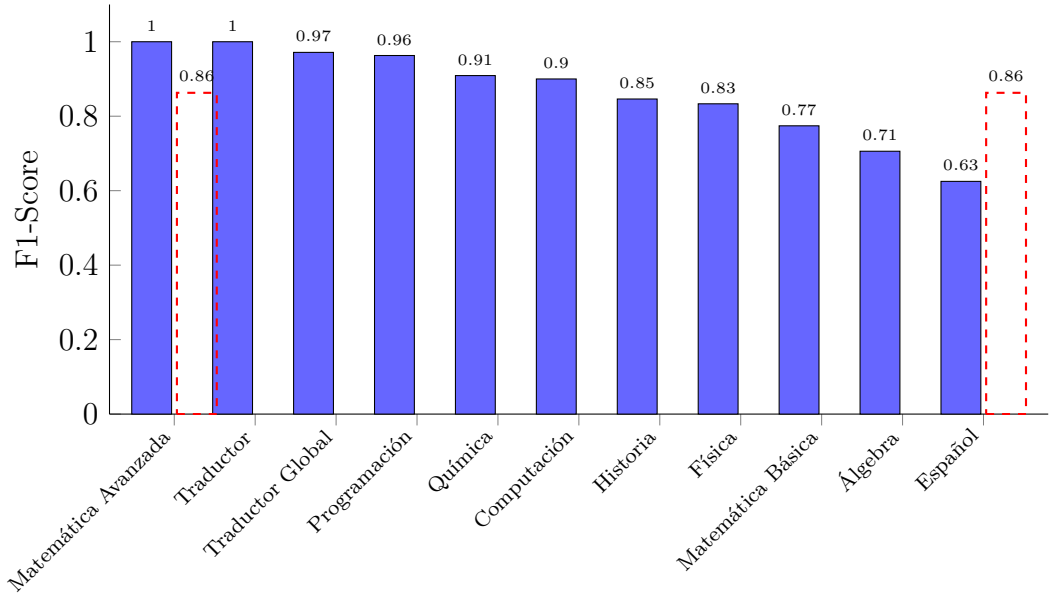


Figura 5.8: F1-Score por asistente especializado — Gemma 2 9B. La línea punteada roja representa la media global (0.8628).

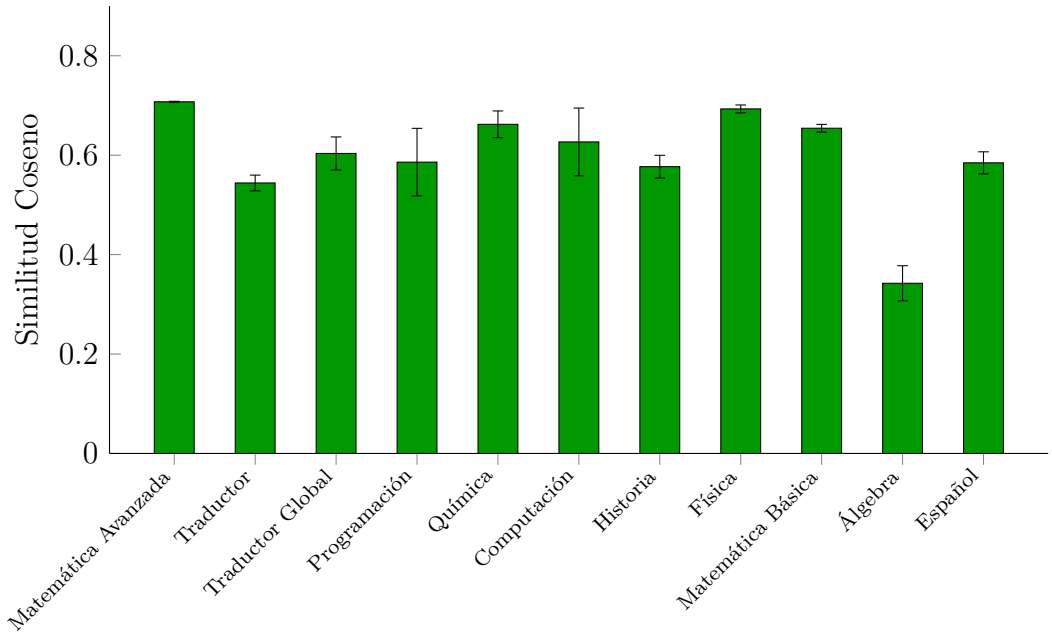


Figura 5.9: Similitud Coseno promedio por asistente especializado — Gemma 2 9B (media ± desviación estándar entre cinco iteraciones).

Con Gemma 2 9B el asistente Matemática Avanzada obtiene una puntuación perfecta (F1-Score = 1.0000) , identificando cuándo una consulta requiere herramientas de cálculo complejo (como integrales). Gemma también demostró una comprensión casi perfecta de las reglas de exclusión mutua de traducción: diferenció exitosamente cuándo usar Traductor (F1-Score = 1.000) y cuándo usar Traductor Global (F1-Score = 0.9714). No obstante, el asistente de Álgebra presentó el menor desempeño

semántico del grupo con una Similitud Coseno preocupantemente baja (0.3423), lo que indica que si bien el modelo clasifica correctamente las consultas de Álgebra, el ensamblaje o las respuestas generadas sufren de degradación de formato o falta de precisión.

5.5.4 Análisis de fallos persistentes

Se detectó una tasa alta de fallos persistentes con coseno = 0 en las cinco iteraciones, donde seis preguntas fallaron sistemáticamente.

Tabla 5.10: Preguntas con fallo persistente — Gemma 2 9B

ID	**Pregunta**	**Asistentes esperados**	**Asistentes invocados**
H06	Diferencia entre Aritmética y Álgebra, operadores new/delete, 20/4 y traducción al portugués.	Álgebra, Programación, Matemática Básica, Traductor Global	Matemática Básica, Programación, Álgebra, Traductor Global
H07	Resume quién fue Albert Einstein, explica qué es la energía cinética, qué es un enlace covalente y cómo funciona el sistema binario basado en transistores.	Historia, Física, Química, Computación	Matemática Básica, Programación, Álgebra, Traductor Global
H11	¿Cuáles son los signos de agrupación en álgebra, qué es una variable de tipo puntero en C++, cuánto es 1/2 + 1/4 y traduce todo al francés?	Álgebra, Programación, Matemática Básica, Traductor Global	Álgebra, Matemática Básica, Programación, Español, Traductor Global

Tabla 5.10: Preguntas con fallo persistente — Gemma 2 9B (continuación).

ID	Pregunta	Asistentes esperados	Asistentes invocados
H19	¿Cuál es la diferencia entre Aritmética y Álgebra, qué es la CPU, cuál es la diferencia entre rapidez y velocidad, y simplifica la fracción 4/8?	Álgebra, Computación, Física, Matemática Básica	Matemática Básica, Computación, Álgebra
H28	¿Qué establece la Primera Ley de Newton, cuáles son las excepciones para omitir los signos de interrogación, qué es la herencia en C++ y traduce todo al alemán?	Física, Español, Programación, Traductor Global	Física, Álgebra, Programación, Español, Traductor Global
H33	¿Qué fue la ENIAC, a qué es igual el cuadrado de la suma de dos cantidades, integra $x^3 - 2x$ y traduce todo al portugués?	Computación, Álgebra, Matemática Avanzada, Traductor Global	Computación, Historia, Matemática Básica, Matemática Avanzada, Español, Traductor Global

Aislando el caso de la pregunta H07 (que es un fallo de invocación o alucinación), las otras cinco preguntas revelan el problema fundamental: cuatro de las seis preguntas fallidas (H06, H11, H28, H33) requerían la participación del Traductor Global.

Esto evidencia que el asistente encargado de recibir las respuestas de tres o cuatro asistentes (cada una con formatos pesados de fórmulas, JSON y texto) experimenta desbordamientos masivos. En el caso de H33, el orquestador invocó a 6 asistentes distintos simultáneamente, lo que sobrecargó al Traductor Global intentando consolidar la historia, las fórmulas de álgebra avanzada y el texto, generando invariablemente un error interno o un formato ininteligible que colapsa la métrica de similitud a 0.0.

5.6 Resultados de la Fase 2: evaluación con Mistral 12B

En esta subsección se analizan los resultados de la Fase 2 utilizando el modelo Mistral 12B como clasificador y enrutador principal del orquestador. Al igual que

en la evaluación anterior, se procesaron las 35 preguntas (ver Anexo I) del conjunto de datos híbrido en 5 iteraciones independientes, operando bajo el parámetro de `temperature=0.0`.

5.6.1 Desempeño global

La Tabla 5.11 resume las métricas del orquestador consolidando las cinco iteraciones de evaluación.

Tabla 5.11: Métricas globales del orquestador — Mistral 12B (media y desviación estándar entre cinco iteraciones)

Métrica	**Media**	**Desv. estándar**
Precisión	0.8876	0.0000
Exhaustividad	0.9014	0.0000
F1-Score	0.8923	0.0000
Similitud Coseno	0.5562	0.0243
Latencia Global (segundos)	485.17	41.20

Los resultados globales revelan una mejora en la capacidad de comprensión lógica de Mistral 12B frente al modelo Gemma 2 9B evaluado en la sección anterior. El F1-Score de ruteo asciende a 0.8923, posicionándose como un desempeño excelente en la clasificación multi-etiqueta. La varianza nula en el F1-Score confirma nuevamente que el enrutamiento es determinista. Sin embargo, a pesar de que el modelo escoge correctamente a los asistentes casi en el 90 % de los casos, la Similitud se mantiene en un nivel moderado (0.5562 ± 0.0243), un valor estadísticamente equivalente al obtenido por modelos más pequeños. Esto sugiere que el cuello de botella del sistema se desplaza del error de ruteo hacia la complejidad de orquestar y formatear múltiples respuestas en paralelo. Adicionalmente, el costo computacional es notable: la Latencia Global promedio casi se duplica, alcanzando los 485.17 segundos por consulta.

5.6.2 Sesgo del *prompt* de clasificación

El análisis detallado del ruteo demuestra que Mistral 12B supera con éxito la dependencia al ejemplo base que limitó de forma considerable el desempeño de Llama 8B. Mistral no replicó ciegamente el conjunto de asistentes de ejemplo, sino que analizó de forma independiente cada consulta.

El hallazgo más destacado de este modelo es su capacidad para seguir instrucciones complejas de exclusión mutua descritas en el *prompt* del sistema. Por ejemplo, el asistente Matemática Avanzada fue invocado exitosamente en todas las consultas que requerían cálculo integral o ecuaciones avanzadas. Del mismo modo, Mistral discriminó correctamente cuándo delegar la tarea al Traductor Global para traducciones de respuesta completa, ignorando al asistente Traductor (cuyo F1-Score fue 0.0000 por no ser requerido en la semántica global de las consultas).

5.6.3 Desempeño por asistente especializado

La Tabla 5.12 y las Figuras 5.10 y 5.11 presentan el desglose de métricas por asistente especializado.

Tabla 5.12: Métricas por asistente especializado — Mistral 12B (ordenado por F1-Score)

Asistente	**F1-Score**	**Sim. Coseno**		**Latencia (*s*)**	
		Media	Desv.	Media	Desv.
Química	0.9565	0.5595	0.0035	46.39	1.97
Computación	0.9524	0.6885	0.0346	39.26	1.71
Traductor Global	0.9444	0.6403	0.0541	90.98	10.13
Física	0.9231	0.6945	0.0031	41.41	1.65
Matemática Básica	0.9231	0.6457	0.0039	37.07	0.95
Programación	0.9231	0.6251	0.0355	116.74	4.28
Historia	0.8889	0.6279	0.0215	42.18	1.60
Álgebra	0.8421	0.5408	0.0292	75.00	8.79
Matemática Avanzada	0.8235	0.7302	0.0174	72.35	6.69
Español	0.7333	0.6061	0.0182	51.43	5.53
Traductor	0.0000	0.0000	0.0000	0.00	0.00

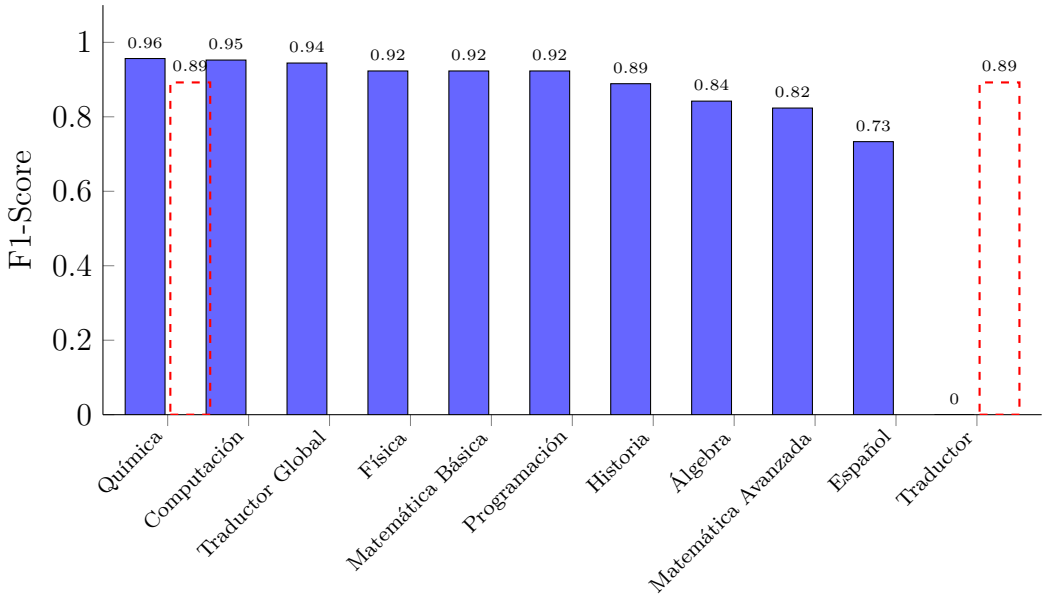


Figura 5.10: F1-Score de ruteo por asistente especializado — Mistral 12B. La línea punteada roja representa la media global (0.8923).

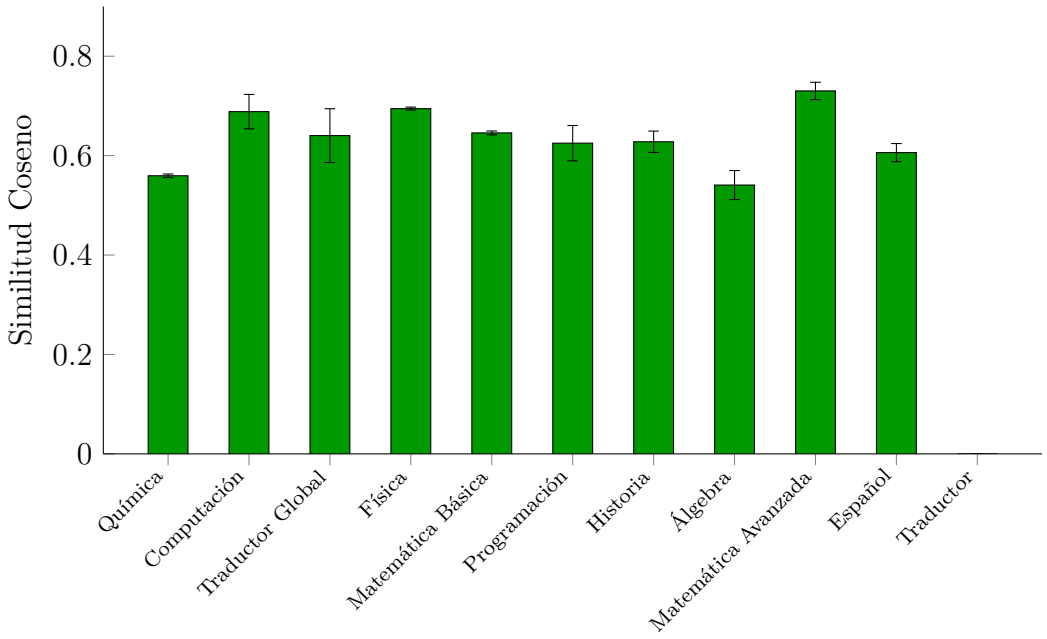


Figura 5.11: Similitud Coseno promedio por asistente especializado — Mistral 12B (media ± desviación estándar entre cinco iteraciones).

El ruteo por asistente es uniforme y preciso. Seis de los once asistentes presentan un F1-Score superior a 0.90, destacando Química (0.9565) y Computación (0.9524). Es vital resaltar el desempeño de Matemática Avanzada, que saltó de 0.0 en el modelo anterior a 0.8235 con Mistral. Además, Matemática Avanzada es el asistente que reporta la mayor Similitud Coseno del ecosistema (0.7302 ± 0.0174), lo que indica que cuando Mistral lo invoca (haciéndolo casi siempre correctamente), las respuestas generadas conservan una alta fidelidad estructural.

Al igual que en otras iteraciones, `programacion` es el asistente con mayor Latencia (116.74 s), pero en esta arquitectura, el Traductor Global también exige un alto coste en tiempo de ejecución (90.98 s con alta desviación estándar), lo cual impacta directamente en los fallos persistentes documentados en 5.6.4.

5.6.4 Análisis de fallos persistentes

Se evaluaron las preguntas que obtuvieron una Similitud Coseno igual a 0.0 de forma ininterrumpida a lo largo de las cinco iteraciones. Se identificaron cuatro casos consistentes: H06, H11, H14 y H18.

Tabla 5.13: Preguntas con fallo persistente — Mistral 12B (continuación).

ID	Pregunta	Asistentes esperados	Asistentes invocados
H06	¿Cuál es la diferencia entre Aritmética y Álgebra, para qué sirven los operadores `new` y `delete` en C++, cuánto es 20/4, y traduce todo el resultado al portugués?	Álgebra, Programación, Matemática Básica, Traductor Global	Matemática Básica, Álgebra, Programación, Traductor Global
H11	¿Cuáles son los signos de agrupación en álgebra, qué es una variable de tipo puntero en C++, cuánto es $1/2 + 1/4$ y traduce todo al francés?	Álgebra, Programación, Matemática Básica, Traductor Global	Programación, Álgebra, Matemática Básica, Traductor Global
H14	¿Cuáles fueron los objetivos de la ONU, cómo se define el concepto de mol, cuánto es $15 + 27$, y traduce todo al portugués?	Historia, Química, Matemática Básica, Traductor Global	Español, Química, Matemática Básica, Traductor Global
H18	¿Qué aportó el judaísmo ideológicamente, cuál es la diferencia entre enlace iónico y covalente, y traduce todo al alemán?	Historia, Química, Traductor Global	Química, Español, Traductor Global

A diferencia del modelo anterior Gemma 2 9B donde los fallos se debían a malas decisiones de orquestación, el análisis de Mistral revela que: en las preguntas H06 y H11, la orquestación de Mistral fue matemáticamente perfecta (el modelo seleccionó el conjunto exacto de cuatro asistentes requeridos), pero aun así el sistema reportó un fallo total (coseno = 0).

El patrón común en las cuatro preguntas es la presencia obligatoria del asistente Traductor Global sumado a una carga de entre tres y cuatro asistentes especialistas. Dado el aumento drástico en la Latencia de Mistral (485 segundos de media), esto

sugiere que el fallo ocurre en la fase de ensamblaje y traducción final. El asistente Traductor recibe una concatenación de tres respuestas técnicas profundas y debe procesarlas en un solo paso, lo que excede la ventana de contexto operativo, provoca un desbordamiento de memoria o corrompe el formato JSON estandarizado, llevando la evaluación de la calidad semántica a cero.

5.7 Resultados de la Fase 2: evaluación con Qwen 2.5 14B

En esta subsección se evalúa el desempeño del orquestador empleando el modelo Qwen 2.5 14B. Al ser el modelo con mayor cantidad de parámetros del estudio, su arquitectura está especialmente optimizada para la asimilación de instrucciones complejas y el formateo estricto de datos. Como en las fases previas, se procesaron las 35 preguntas (ver Anexo I) del conjunto híbrido en cinco iteraciones bajo un escenario determinista.

5.7.1 Desempeño global

La Tabla 5.14 resume las métricas macrocéntricas del sistema utilizando Qwen como enrutador principal.

Tabla 5.14: Métricas globales del orquestador — Qwen 2.5 14B (media y desviación estándar entre cinco iteraciones)

Métrica	**Media**	**Desv. estándar**
Precisión	0.9538	0.0000
Exhaustividad	0.9538	0.0000
F1-Score	0.9538	0.0000
Similitud Coseno	0.6339	0.0246
Latencia Global (segundos)	444.46	35.14

Los resultados globales posicionan a Qwen 14B como el modelo de mejor desempeño dentro de este conjunto experimental. El F1-Score alcanza un impresionante 0.9538, superando significativamente a Llama 8B, Gemma 9B y Mistral 12B. Además, se observa un salto cualitativo vital: la Similitud Coseno asciende a 0.6339, rompiendo el techo de ≈ 0.55 en el que se habían estancado las arquitecturas anteriores.

Este incremento en la similitud semántica general confirma que Qwen no solo es altamente preciso para seleccionar a los asistentes correctos, sino que posee una ventana de contexto mucho más robusta para procesar, asimilar y retornar el ensamblaje de las respuestas sin sufrir una degradación de información severa. Este procesamiento pesado, sin embargo, conlleva un costo: una Latencia Global promedio de 444.46 segundos, alineada con el peso computacional de manejar 14 mil millones de parámetros.

5.7.2 Desempeño por asistente especializado

El desglose por asistente (Tabla 5.15 y Figuras 5.12 y 5.13) muestra un panorama de altísima fiabilidad en la clasificación de dominios.

Tabla 5.15: Métricas por asistente especializado — Qwen 2.5 14B (ordenado por F1-Score)

Asistente	**F1-Score**	**Sim. Coseno**		**Latencia (*s*)**	
		Media	Desv.	Media	Desv.
Física	1.0000	0.6976	0.0039	35.63	0.97
Química	1.0000	0.6612	0.0055	47.25	5.01
Traductor global	1.0000	0.6784	0.0330	121.13	6.49
Historia	0.9655	0.6386	0.0087	32.95	2.62
Programación	0.9600	0.5877	0.0790	117.68	4.82
Español	0.9565	0.6087	0.0248	33.54	1.15
Computación	0.9524	0.6678	0.0369	41.15	2.62
Matemática Avanzada	0.9524	0.6966	0.0307	78.15	3.86
Matemática Básica	0.9231	0.6374	0.0269	37.48	1.82
Álgebra	0.7500	0.7115	0.0085	59.40	2.82
Traductor	0.6667	0.6912	0.0004	9.06	0.11

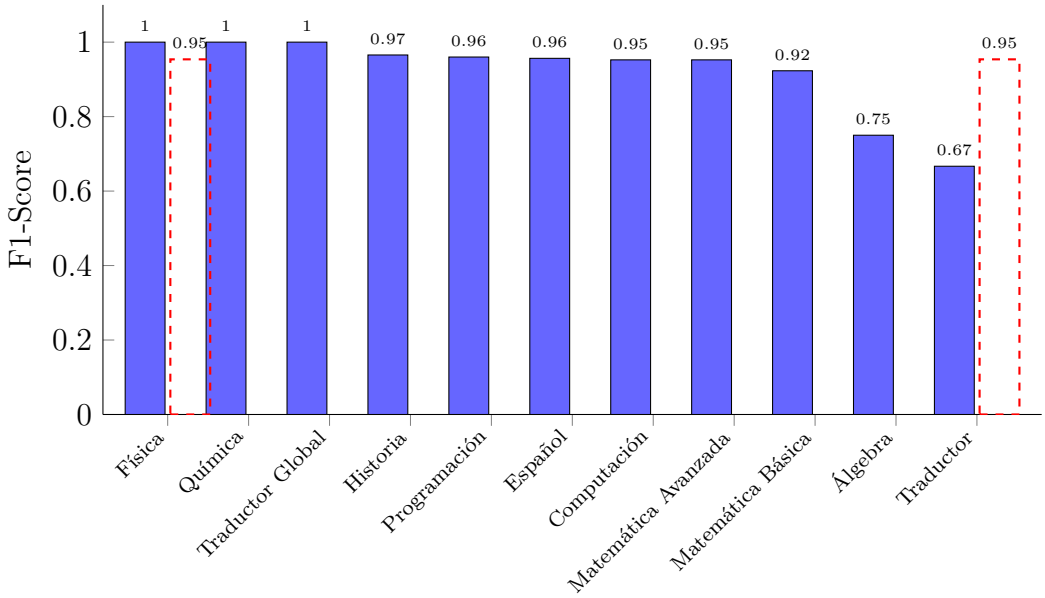


Figura 5.12: F1-Score por asistente — Qwen 2.5 14B. La línea punteada roja representa la media global (0.9538).

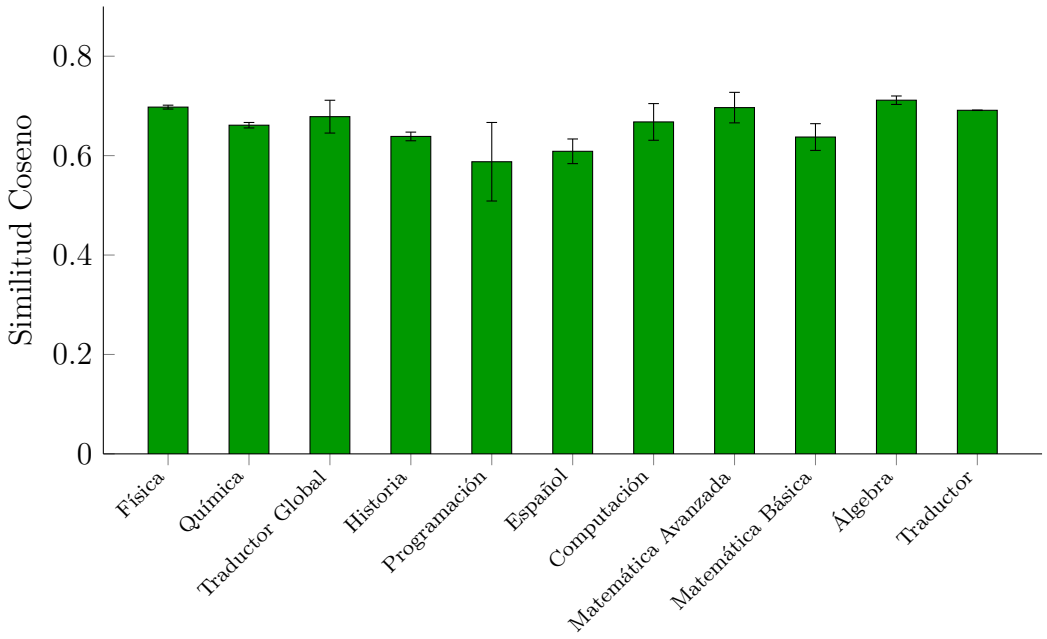


Figura 5.13: Similitud Coseno promedio por asistente especializado — Qwen 2.5 14B (media ± desviación estándar entre cinco iteraciones).

Nueve de los once asistentes disponibles lograron un F1-Score superior a 0.92. Qwen obtuvo (F1-Score = 1.0000) estadísticamente perfecto al invocar a los asistentes de Física, Química y el Traductor Global. Resulta sumamente interesante observar el caso del asistente de Álgebra: si bien presenta el F1-Score de clasificación más bajo (0.7500), obtiene irónicamente la Similitud Coseno más alta de todo el ecosistema (0.7115). Esto significa que aunque Qwen 14B dudó u omitió llamar a Álgebra en el 25 % de los escenarios esperados, cuando sí lo invocó, logró ensamblar su respuesta con una fidelidad matemática y de formato muy superior a la que lograron Mistral o Gemma.

5.7.3 Análisis de fallos persistentes

Uno de los hallazgos más relevantes en la evaluación de Qwen 14B es su estabilidad estructural en la fase final de síntesis. Al analizar los registros en búsqueda de caídas críticas, los datos revelaron que, no se registraron fallos persistentes (similitud Coseno = 0.0) en ninguna de las cinco iteraciones de prueba. Las preguntas altamente complejas como H06, H11, H14 y H33 —las cuales provocaron desbordamientos de memoria y colapsos sistemáticos en Llama, Mistral y Gemma al exigir la traducción de cuatro respuestas técnicas masivas concatenadas— fueron procesadas exitosamente por Qwen. El asistente Traductor Global registró una Latencia alta (121.13 segundos), pero logró sostener la ventana de contexto, decodificar el formato OFP de múltiples especialistas y emitir una respuesta consolidada válida. Este resultado es consistente con la hipótesis de que la mayor capacidad de seguimiento de contexto extenso de Qwen 2.5 contribuye a evitar el colapso de orquestación en cargas de trabajo extremas, aunque una atribución causal definitiva excedería el alcance de esta evaluación.

5.8 Comparación general entre modelos

Las secciones anteriores presentaron los resultados de la Fase 2 de forma independiente para cada uno de los cuatro modelos evaluados como motor de orquestación. Esta sección consolida dichos resultados en una comparación transversal directa, con dos propósitos complementarios: en primer lugar, contrastar las métricas globales, calidad semántica y Latencia entre modelos LLM (Sección 5.8.1); en segundo lugar, unificar el registro de fallos persistentes documentado por separado en cada subsección anterior, con el fin de distinguir los fallos atribuibles a la arquitectura específica de un modelo de aquellos que se manifiestan de forma transversal, independientemente del modelo LLM utilizado como orquestador (Sección 5.8.2).

5.8.1 Métricas globales comparadas

La Tabla 5.16 resume las cuatro métricas principales reportadas individualmente en las Tablas 5.5, 5.11, 5.8 y 5.14, ordenadas de forma ascendente según el número de parámetros del modelo evaluado.

Tabla 5.16: Comparación de métricas globales entre los cuatro modelos evaluados

Modelo	**Parám.**	**F1-Score**	**Sim. Coseno**	**Latencia (*s*)**
Llama 3.1	8B	0.4892	0.5549 ± 0.0259	277.30 ± 7.01
Gemma 2	9B	0.8628	0.5522 ± 0.0117	436.45 ± 21.50
Mistral NeMo	12B	0.8923	0.5562 ± 0.0243	485.17 ± 41.20
Qwen 2.5	14B	0.9538	0.6339 ± 0.0246	444.46 ± 35.14

Dos patrones se distinguen al leer la tabla por columnas. Primero, el F1-Score crece de forma casi monótona con el número de parámetros, desde 0.4892 en Llama 3.1 8B hasta 0.9538 en Qwen 2.5 14B, un incremento de 0.4646 puntos (95 % relativo) a lo largo del rango de tamaños evaluado. El mayor salto absoluto ocurre entre los 8B y los 9B de parámetros (+0.3736), mientras que los incrementos entre 9B y 12B (+0.0295) y entre 12B y 14B (+0.0615) son considerablemente más moderados, lo que sugiere una ganancia marginal decreciente a partir de los 9 mil millones de parámetros. Esta tendencia contrasta con la observada en la Fase 1 (Tabla 5.4), donde el F1-Score medio no guardaba relación monótona con el tamaño del modelo.

Segundo, la Similitud Coseno de extermo a extremo se mantiene prácticamente estancada en un rango de 0.5522 a 0.5562 para Llama, Gemma y Mistral –una diferencia de apenas 0.0040 puntos a pesar de que el F1-Score varía entre estos tres modelos en hasta 0.43 puntos–, y solo Qwen 2.5 14B rompe este techo, alcanzando 0.6339 (+0.078 respecto al promedio de los tres modelos anteriores). En cuanto al costo computacional, la Latencia Global no sigue el mismo ordenamiento que el tamaño del modelo: Llama 8B es, como es esperable, el más rápido (277.30 s), pero Qwen 14B (444.46 s) resulta más rápido que Mistral 12B (485.17 s) a pesar de tener más parámetros, mientras que Gemma 9B (436.45 s) es el segundo más rápido del

grupo.

5.8.2 Análisis unificado de fallos persistentes

Las Secciones 5.4.4, 5.6.4, 5.5.4 y 5.7.3 reportaron, de forma independiente para cada modelo, las preguntas del conjunto híbrido cuya Similitud Coseno fue igual a 0.0 de manera ininterrumpida en las cinco iteraciones de evaluación. Reportados de forma aislada, estos registros no permiten distinguir si un fallo persistente depende de las particularidades del modelo LLM que actuó como orquestador o si, por el contrario, se origina en un componente del sistema común a todos los modelos evaluados. Para resolver esta ambigüedad, la Tabla 5.17 consolida los cuatro registros en una sola matriz pregunta–modelo, indicando con ✗ las combinaciones en las que se presentó fallo persistente y con – aquellas en las que la pregunta fue resuelta exitosamente.

Tabla 5.17: Matriz unificada de fallos persistentes por pregunta y modelo

ID	**Asistentes**	**Llama**	**Mistral**	**Gemma**	**Qwen**	**Total**
H06	4	✗	✗	✗	–	3/4
H11	4	–	✗	✗	–	2/4
H07	4	–	–	✗	–	1/4
H09	4	✗	–	–	–	1/4
H13	4	✗	–	–	–	1/4
H14	4	–	✗	–	–	1/4
H18	3	–	✗	–	–	1/4
H19	4	–	–	✗	–	1/4
H22	3	✗	–	–	–	1/4
H28	4	–	–	✗	–	1/4
H33	4	–	–	✗	–	1/4
Total por modelo		4	4	6	0	11/35

De las 35 preguntas del conjunto híbrido, 11 (31 %) registraron al menos un fallo persistente en alguno de los cuatro modelos evaluados. La columna “Total” confirma que la gran mayoría de estos casos – nueve de las 11 preguntas (82 %)– fallan en una única arquitectura, lo que indica que su causa es específica del comportamiento de ese modelo en particular. Únicamente dos preguntas, H06 y H11, fallan en más de un modelo (3/4 y 2/4 modelos, respectivamente) y ambas comparten un patrón común: requieren la invocación del asistente Traductor Global junto con la coordinación simultánea de cuatro asistentes especialistas. Gemma 2 9B presenta el mayor número de fallos persistentes del grupo (6 de 35 preguntas), mientras que Qwen 2.5 14B es el único modelo sin ningún fallo persistente registrado en las cinco iteraciones.

En conjunto, las Tablas 5.16 y 5.17 muestran que el desempeño del orquestador, la calidad semántica y la resiliencia ante fallos de ensamblaje no escalan de forma idéntica

con el tamaño del modelo. Las implicaciones de esta disociación –particularmente respecto al papel del asistente Traductor Global en los dos fallos transversales– se desarrollan con mayor profundidad en la Sección 5.9.

5.9 Discusión de resultados

Esta sección integra los hallazgos descritos a lo largo de las Fases 1 y 2 para discutir sus implicaciones metodológicas y de diseño, más allá de la simple descripción de los datos presentados en las tablas y figuras anteriores.

5.9.1 Del rendimiento individual al desempeño global del sistema

Un contraste revelador surge al comparar la Similitud Coseno obtenida por los asistentes en aislamiento (Fase 1, Tabla 5.4) con la Similitud Coseno del sistema completo (Fase 2, Tabla 5.16). En la Fase 1, los cuatro modelos generadores alcanzan una Similitud Coseno promedio de 0.728 a 0.735 al responder de forma aislada. En la Fase 2, sin embargo, ningún modelo orquestador supera una Similitud Coseno de 0.634, y los tres modelos LLM con peor desempeño de ensamblaje (Llama, Gemma, Mistral) se estancan en un rango de 0.552 a 0.556 —una caída de entre 0.17 y 0.18 puntos respecto al promedio individual de la Fase 1.

Esta brecha indica que, para el sistema evaluado en este estudio, la degradación observada al integrar los asistentes bajo el orquestador no se origina en la calidad intrínseca de los asistentes especialistas —que ya fue establecida como alta y estable en la Fase 1—, sino en el proceso de orquestación mismo: la segmentación de la consulta, la ejecución concurrente de múltiples asistentes y, sobre todo, el ensamblaje final de sus respuestas bajo el protocolo OFP. Este hallazgo constituye uno de los aportes centrales de esta evaluación: sugiere que, en sistemas de interoperabilidad multi-asistente similares al evaluado, el principal cuello de botella puede no residir en la competencia individual de sus componentes, sino en la capa de coordinación que los integra.

5.9.2 El umbral de 0.75 como fuente de error de medición, no de desempeño real

Los resultados de la Fase 1 muestran dos casos de descalibración opuestos entre el F1-Score binario y la Similitud Coseno continua (Figura 5.4). En un extremo, Historia obtiene el F1-Score más bajo del estudio (Figura 5.1) pese a que su Similitud Coseno (0.691–0.720) se ubica apenas por debajo del umbral de clasificación, lo que indica que sus respuestas son semánticamente cercanas a la referencia y que el corte binario en 0.75 las penaliza como fallo total. En el extremo opuesto, Matemática Básica y Matemática Avanzada obtienen dos de los F1-Score más altos del estudio (0.889–0.966 y 0.846–1.000) a pesar de registrar Similitud Coseno media más baja de

los diez asistentes (0.490–0.619): aquí el umbral clasifica como acierto a respuestas cuya cercanía semántica continua con la referencia es, en términos absolutos, baja.

Ambos casos sugieren que el umbral fijo de 0.75 no es igualmente discriminante para todos los dominios evaluados, posiblemente porque la dispersión natural de los *embeddings* varía según la longitud y la estructura léxica de la respuesta esperada (prosa histórica extensa frente a expresiones matemáticas o numéricas breves). Esto no invalida el umbral como criterio general, pero sí indica que su calibración por dominio –o el uso complementario de la métrica continua de Similitud Coseno junto al F1-Score binario, como se hizo en este estudio– es preferible a reportar un único valor agregado, ya que el umbral fijo puede ocultar tanto falsos fallos (Historia) como falsos aciertos (Matemática) si se interpreta de forma aislada.

5.9.3 La escala como habilidad emergente de orquestación, no de especialización

Al contrastar los promedios globales de la Fase 1 (Tabla 5.4) con la comparación de F1-Score de la Fase 2 (Tabla 5.16) se observa un patrón cualitativamente distinto entre ambas fases. En la Fase 1, el tamaño del modelo no guarda relación monótona con el desempeño: Gemma 9B obtiene el F1-Score medio más alto (0.879), por encima de Qwen 14B (0.870) y Mistral 12B (0.868), lo que indica que, para tareas de asistentes único sin necesidad de planificación ni coordinación, el rango de 8 a 14 mil millones de parámetros evaluado no impone una diferencia de capacidad determinante; las diferencias entre asistentes especialistas y su diseño de *prompt* resultan considerablemente mayores que las diferencias entre modelos.

En la Fase 2, en cambio, el F1-Score crece de forma casi monótona con el número de parámetros ($0.49 \rightarrow 0.86 \rightarrow 0.89 \rightarrow 0.95$), con un salto especialmente marcado entre 8B y 9B. Esta divergencia entre fases sugiere que la capacidad de descomponer una consulta compleja, asignarla a múltiples asistentes y respetar reglas de exclusión mutua constituye una habilidad emergente que se beneficia de la escala del modelo, mientras que la comprensión y respuesta de una consulta de dominio aislado no depende, dentro del rango de tamaños evaluado, del número de parámetros sino del diseño del asistente especialista individual. Esto tiene una implicación práctica directa para el diseño de sistemas similares: invertir en un modelo de mayor escala solo aporta beneficio claro en el componente de *enrutamiento*, no en los componentes especialistas, donde un modelo más pequeño y bien afinado a su dominio (como se observó con `gemma2:2b` o `qwen2.5:3b` en varios asistentes de la Fase 1) resulta computacionalmente más eficiente sin sacrificar calidad.

5.9.4 El cuello de botella estructural de la fase de ensamblaje

El análisis por modelo de la Fase 2 reveló dos modos de fallo cualitativamente distintos. El primero, observado principalmente en Llama 3.1 8B, es un fallo: el sesgo de anclaje al ejemplo base del *prompt* de clasificación impide que el modelo razone independientemente sobre el contenido semántico de cada consulta, lo que se tradujo en la exclusión total del asistente Matemática Avanzada en las cinco iteraciones.

Mistral 12B, Gemma 2 9B y Qwen 2.5 14B superan este sesgo y alcanzan F1-Score de 0.86 a 0.95.

El segundo modo de fallo, sin embargo, persiste incluso cuando el ruteo es correcto: la matriz unificada de fallos persistentes (Tabla 5.17) muestra que dos preguntas (H06 y H11) –ambas con cuatro asistentes especialistas y el asistente Traductor Global obligatorio– colapsan en más de una arquitectura, incluyendo a Mistral 12B en un escenario donde su ruteo fue matemáticamente perfecto. Esto constituye evidencia de un fallo estructural del sistema, independiente del modelo usado como enrutador: la fase de ensamblaje y traducción final colapsa bajo la carga de contexto cuando debe consolidar respuestas extensas de tres o cuatro especialistas concurrentes, ya sea por desbordamiento de la ventana de contexto, por *timeout* o por corrupción del formato OFP estandarizado. Las restantes nueve preguntas con fallo persistente son atribuibles a errores de ruteo específicos de cada arquitectura —anclaje al ejemplo base del *prompt* en Llama, o confusión y sobre-invocación de asistentes en Gemma y Mistral— y no al cuello de botella estructural.

Qwen 2.5 14B es el único modelo LLM evaluado que no presenta ningún fallo persistente, lo que sugiere que su ventana de contexto y capacidad de seguimiento de dependencias largas le permiten sostener el ensamblaje de respuestas múltiples sin degradar el formato de salida, a costa de una Latencia Global mayor (444.46 s en promedio) que en algunos casos resulta menor incluso a la de Mistral 12B (485.17 s), pese a su mayor tamaño.

5.9.5 Conclusiones de la evaluación de la Fase 1 y la Fase2

En conjunto, los resultados de ambas fases indican que la elección de un modelo orquestador para un sistema de interoperabilidad multi-asistente bajo el protocolo OFP no puede basarse únicamente en el desempeño individual de los asistentes (donde la escala no es determinante) sino en su capacidad de razonamiento de ruteo y, sobre todo, en su resiliencia estructural ante el ensamblaje de respuestas concurrentes extensas. De los cuatro modelos evaluados, Qwen 2.5 14B es el único que combina un F1-Score casi perfecto (0.9538), el mayor salto en Similitud Coseno (0.6339) y ausencia total de fallos persistentes, lo que lo posiciona como la arquitectura más adecuada para escenarios de orquestación de alta carga cognitiva, con la salvedad de que su tiempo de inferencia (superior a 7 minutos por consulta híbrida) representa una limitación relevante para implementaciones que requieran respuesta en tiempo real.

5.10 Evaluación de la carga cognitiva

La evaluación precedente caracterizó el desempeño del modelo de interoperabilidad desde una perspectiva estrictamente computacional. Sin embargo, las variables de rendimiento técnico no responden de forma directa a la pregunta que originó esta investigación: si la fragmentación entre asistentes virtuales heterogéneos, al ser resuelta mediante una orquestación centralizada, disminuye de forma efectiva el esfuerzo mental

de los usuarios. Conviene precisar desde ahora qué mide exactamente esta evaluación: no la capacidad de cada asistente especialista para resolver su parte de la consulta —eso ya se validó en las secciones anteriores—, sino el costo de coordinarlos: decidir a quién preguntar, en qué orden, y cómo ensamblar sus respuestas en un resultado único. Por lo tanto, este análisis complementa la perspectiva de infraestructura con una medición subjetiva de la carga cognitiva percibida, empleando el cuestionario NASA-TLX como instrumento estandarizado. A continuación, se detalla el diseño de la evaluación, el procedimiento seguido y el análisis de los resultados obtenidos en contraste con los hallazgos computacionales del sistema.

5.10.1 Cuestionario RTLX: diseño de la evaluación

La evaluación de carga cognitiva se diseñó como un estudio de medidas repetidas de dos condiciones, usando como modelo LLM Qwen 2.5 14B — el modelo con mejor desempeño global establecido en la Sección 5.9.5. Para medir la carga se utilizó el cuestionario NASA-TLX, en su variante RTLX (sin la fase de ponderación por comparaciones pareadas; véase la Sección 5.2):

- **Condición A (flujo descentralizado):** el participante recibe los puntos de acceso individuales de los asistentes especialistas involucrados en la consulta y debe identificar a qué asistente corresponde cada fragmento de la consulta, enviarlos por separado, y consolidar manualmente las respuestas —incluida, en su caso, la traducción del resultado final— en un único documento.
- **Condición B (flujo orquestado):** el participante envía la consulta completa una sola vez al punto de acceso único del orquestador, el cual gestiona internamente la segmentación, el ruteo, la ejecución concurrente y el ensamblaje bajo el protocolo OFP.

Ambas condiciones se administraron con la misma herramienta cliente mínima, construida solo para esta evaluación: envía las peticiones HTTP y muestra la respuesta de cada asistente en un formato uniforme de lectura. Usar la misma herramienta en ambas condiciones aísla la variable que realmente interesa —cuántos pasos de orquestación debe hacer el participante a mano— de cualquier diferencia de usabilidad de la herramienta en sí, y permite reclutar participantes sin experiencia técnica previa en clientes REST.

5.10.2 Participantes y procedimiento

Se reclutó una muestra de $n = 16$ participantes voluntarios con alfabetización digital básica (uso de teclado, lectura de texto en pantalla), sin requerir formación en programación. Se utilizaron dos consultas híbridas de dificultad equivalente, H06 y H11, ambas requieren la coordinación de cuatro asistentes y la traducción de la síntesis final al inglés. Para que la dificultad de una pregunta no se confunda con el efecto de la condición, el diseño contrabalancea a la vez dos factores —el orden de las condiciones y qué pregunta se responde en cada una— mediante cuatro grupos de tamaño equivalente ($n = 4$ cada uno):

Tabla 5.18: Esquema de contrabalanceo de orden y asignación pregunta–condición

Grupo	Primer bloque	Segundo bloque
1	H06 / Condición A	H11 / Condición B
2	H11 / Condición A	H06 / Condición B
3	H06 / Condición B	H11 / Condición A
4	H11 / Condición B	H06 / Condición A

Después de completar cada bloque, el participante respondió el cuestionario RTLX correspondiente. Adicionalmente, se registraron dos medidas objetivas: el tiempo total de finalización de la tarea y si la respuesta final coincidía con la referencia de validación. Esto permitió contrastar la percepción subjetiva del participante sobre su propio rendimiento con un criterio de éxito independiente.

5.10.3 Resultados

Antes de entrar en los números agregados, vale la pena detenerse en un caso concreto que resume el fenómeno completo. El participante P07 resolvió la consulta H11 en el flujo manual: coordinó él mismo a los cuatro asistentes especialistas, pero al final se le olvidó pasar la respuesta unificada por el adaptador de traducción —un paso más entre muchos que él debía recordar ejecutar—. El resultado quedó incorrecto y su RTLX de ese bloque llegó a 63.33. Poco después, el mismo participante resolvió la consulta H06 con el orquestador: el sistema se encargó de la traducción sin que él tuviera que acordarse de nada, y su RTLX cayó a 16.67. Como quedó registrado en sus propias observaciones durante la sesión, el orquestador resolvió lo que él olvidó hacer manual. El asistente Traductor no cambió entre una condición y otra —el traductor sabía traducir igual de bien en ambos casos—; lo que cambió fue quién tenía que acordarse de invocarlo. Este contraste, con la misma persona y el mismo tipo de tarea en ambas condiciones, es justamente el patrón que se repite de forma sistemática en los datos agregados que siguen.

Tabla 5.19: Resultados globales del cuestionario RTLX y métricas operativas: flujo descentralizado con respecto a flujo orquestado ($n = 16$).

Métrica / Subescala	Condición A (Manual)	Condición B (Orquestador)
Demanda Mental	71.25 ± 11.47	22.50 ± 7.75
Demanda Física	25.00 ± 6.32	8.13 ± 4.03
Demanda Temporal	63.75 ± 11.47	68.75 ± 10.25
Rendimiento*	20.63 ± 22.05	14.38 ± 31.83
Esfuerzo	77.50 ± 12.38	18.75 ± 6.19
Frustración	60.00 ± 17.51	25.63 ± 27.80
RTLX Global	**53.02 ± 9.47**	**26.35 ± 13.08**
Tiempo de tarea (min)	9.25 ± 2.36	7.67 ± 0.61
Tasa de acierto	13 / 16	12 / 16

*Rendimiento en escala invertida: 0 = Éxito total, 100 = Fracaso total.

Los resultados muestran un contraste claro entre condiciones. La Demanda Mental, el Esfuerzo y la Frustración bajan a menos de la mitad con el flujo orquestado, lo que indica que el participante percibe mucho menos carga mental al no tener que coordinar los asistentes manualmente. La Demanda Temporal, en cambio, sube un poco en la Condición B (68.75 ± 10.25 frente a 63.75 ± 11.47 en A), lo cual tiene sentido si se recuerda que Qwen 2.5 14B tarda más de siete minutos por consulta (Sección 5.9.5): el participante espera más tiempo, aunque haga menos trabajo.

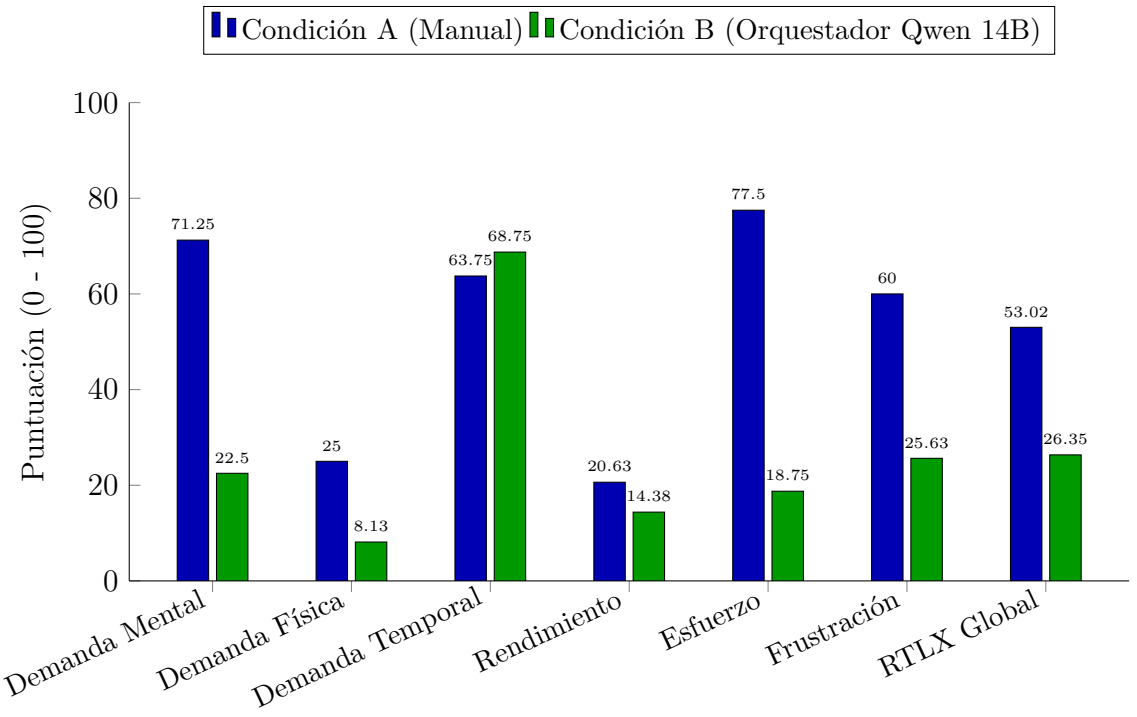


Figura 5.14: Comparativa de la carga cognitiva percibida (RTLX) entre el flujo manual y el flujo orquestado. Valores más bajos indican menor carga o mejor percepción de éxito.

En cuanto a la tasa de efectividad, la Condición A (flujo manual) registró un total de 13/16 soluciones correctas, mientras que la Condición B (flujo orquestado) alcanzó 12/16 aciertos. A pesar de la similitud cuantitativa en los resultados brutos, el análisis cualitativo de las observaciones revela que la naturaleza de los errores difiere sustancialmente entre ambos enfoques.

En la Condición A, las tres soluciones incorrectas se asociaron de manera directa a la carga procesal impuesta por el método de interacción: una de ellas —el caso de P07 descrito al inicio de esta sección— fue consecuencia de una omisión procedimental al olvidar transferir la respuesta unificada por el adaptador de traducción; otra obedeció a un sesgo metodológico donde el usuario intentó resolver la sub-tarea analítica de forma mental eludiendo la consulta al asistente experto correspondiente; y la tercera se derivó de una imprecisión en el desarrollo secuencial estándar bajo la fatiga del flujo manual. En los tres casos, el asistente correcto para resolver la sub-tarea existía y funcionaba: el error no ocurrió porque un asistente respondiera mal, sino porque coordinar manualmente varios asistentes deja espacio para olvidar un paso, atajar uno de ellos, o perder el hilo del procedimiento.

Por otro lado, en la Condición B los cuatro fallos se segmentaron en dos categorías específicas, y ninguna de ellas tiene que ver con que el participante se equivocara al coordinar: fallas de infraestructura tecnológica y factores de tolerancia conductual. Dos de las soluciones incorrectas correspondieron a colapsos críticos de red en tiempo de ejecución, específicamente un desbordamiento por tiempo de espera (*timeout*) en el servicio de traducción global y un fallo de comunicación en el nodo del orquestador central. En contraste, las dos incorrecciones restantes ocurrieron en ejecuciones

donde el sistema no registró anomalías de red, sino que los usuarios interrumpieron prematuramente el ciclo de espera debido a la impaciencia inducida por los tiempos de inferencia locales.

Este comportamiento evidencia un cambio de naturaleza en el error, no solo de frecuencia: los errores en el flujo manual están fuertemente correlacionados con omisiones operativas o fatiga cognitiva del participante —es decir, con el trabajo de coordinación que el usuario tenía que hacer—, mientras que los fallos en el entorno automatizado se dividen entre la robustez de la infraestructura distribuida y las expectativas de Latencia del usuario final —factores externos a la coordinación, que ya no depende de la persona—.

5.10.4 Análisis estadístico

Para confirmar que los datos anteriores no son un efecto de azar propio de una muestra de 16 personas, se compararon las seis subescalas y el índice global entre condiciones mediante la prueba de Wilcoxon de rangos con signo [109], la alternativa no paramétrica a la prueba t pareada. Se optó directamente por esta prueba, sin asumir normalidad, por dos razones: el tamaño de la muestra ($n = 16$) es demasiado reducido para evaluar el supuesto de normalidad con potencia estadística adecuada, y las subescalas del NASA-TLX, aunque se registran en una escala continua de 0 a 100, son de naturaleza ordinal en su interpretación —reflejan una percepción subjetiva ordenada, no una magnitud física medible con precisión—. Un estadístico no paramétrico resulta, en consecuencia, más conservador y apropiado para este tipo de dato [110].

La Figura 5.15 muestra la media y la desviación estándar de cada subescala por condición (los mismos valores de la Tabla 5.19, en gráfico). Se ven tres patrones claros: en Demanda Mental, Demanda Física y Esfuerzo, ambas condiciones están muy separadas, sin superposición entre sus rangos de variación; en Demanda Temporal, la diferencia es mínima; y en Rendimiento y Frustración, la Condición B tiene mucha más dispersión que la A —producto de las dos sesiones con falla de *timeout* ya mencionadas—, tan marcada en el caso de Frustración que ahí sí llegan a solaparse los rangos de ambas condiciones, aunque el promedio de cada una sigue siendo claramente distinto.

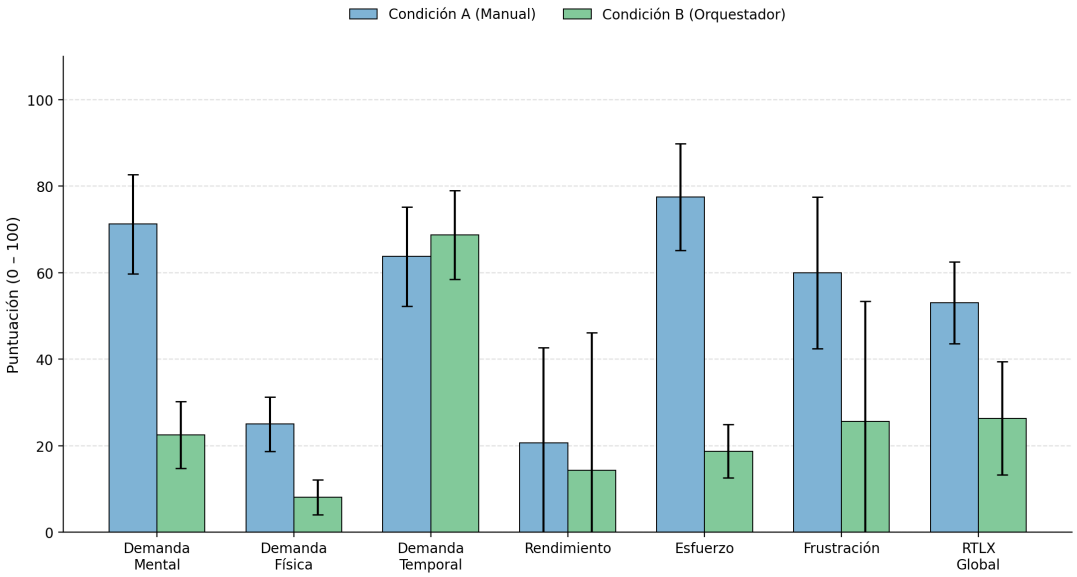


Figura 5.15: Media y desviación estándar por subescala, Condición A (Manual) contra Condición B (Orquestador).

La Figura 5.16 ilustra además el caso del RTLX global: la mayoría de los participantes mejora con el flujo orquestado (diferencias negativas, agrupadas entre -50 y -10), salvo dos valores atípicos positivos que corresponden a las dos sesiones con falla de *timeout* ya mencionadas. Esta asimetría —la mayoría de las diferencias concentradas en un rango, con un par de valores atípicos que se alejan en dirección opuesta— es precisamente el tipo de distribución frente a la cual una prueba no paramétrica como Wilcoxon resulta más robusta que una prueba t, dado que esta última es sensible a valores atípicos y asume simetría en la distribución de las diferencias.

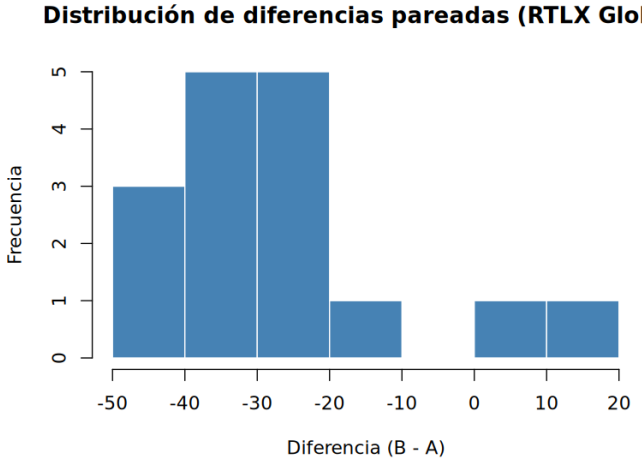


Figura 5.16: Distribución de las diferencias pareadas de RTLX global (B − A).

5.10.5 Prueba de Wilcoxon de rangos con signo

La pregunta que resuelve esta prueba es sencilla: la caída que se observa en la Tabla 5.19 y en la Figura 5.14, ¿es una diferencia real y consistente entre condiciones, o pudo haber surgido por azar con una muestra de apenas 16 personas? Para responderlo se aplicó la prueba de Wilcoxon de rangos con signo [109] a cada variable, contrastando H_0 (no hay diferencia entre condiciones) contra H_1 (sí la hay), con un umbral de significancia de $p < 0.05$. La Tabla 5.20 añade además el tamaño de efecto r [111]: mientras el valor p dice si la diferencia es estadísticamente significativa, r dice qué tan grande es —dos diferencias pueden ser igualmente significativas y, aun así, una decisiva y la otra apenas perceptible.

Antes de leer la tabla conviene precisar qué mide cada columna. La variable V es el estadístico de la prueba de Wilcoxon: la suma de los rangos de las diferencias con signo positivo entre pares (cuanto más cercano a 0, más consistente es la caída en la misma dirección para casi todos los participantes).La variable Z es el estadístico estandarizado que aproxima V a una distribución normal, y su signo indica la dirección del cambio —negativo cuando la Condición B (el orquestador) presenta valores menores que la Condición A—. La variable p es la probabilidad de observar una diferencia igual o más extrema que la registrada si H_0 fuera cierta, i.e., el nivel de significancia de la prueba. r es el tamaño de efecto no paramétrico asociado, calculado como $r = Z/\sqrt{n}$, y a diferencia de p no depende del tamaño muestral, por lo que permite comparar la magnitud del cambio entre variables. Por último, la columna Efecto traduce ese valor de r a una categoría cualitativa según los umbrales de [111] ($r \approx 0.10$ pequeño, $r \approx 0.30$ mediano, $r \approx 0.50$ grande).

Tabla 5.20: Prueba de Wilcoxon de rangos con signo, Condición B vs. Condición A ($n = 16$).

Variable	V	Z	p	r	Efecto	Decisión
Demanda Mental	0.0	-3.51	0.0005	-0.88	Grande	Se rechaza H_0
Demanda Física	0.0	-3.58	0.0003	-0.90	Grande	Se rechaza H_0
Demanda Temporal	38.0	$+1.03$	0.303	0.26	Pequeño	No se rechaza H_0
Rendimiento	29.0	-1.76	0.078	-0.44	Mediano	No se rechaza H_0
Esfuerzo	0.0	-3.54	0.0004	-0.89	Grande	Se rechaza H_0
Frustración	18.5	-2.54	0.011	-0.64	Grande	Se rechaza H_0
RTLX global	4.5	-3.26	0.0011	-0.82	Grande	Se rechaza H_0
Tiempo de tarea (min)	29.0	-2.00	0.046	-0.50	Grande	Se rechaza H_0

En términos simples: cinco de las ocho variables —Demanda Mental, Demanda Física, Esfuerzo, Frustración y RTLX global— caen de forma clara, consistente y con un tamaño de efecto grande, siempre a favor del orquestador; esa caída no es casualidad estadística. El tiempo de tarea también resulta significativo ($p = 0.046$), aunque con un efecto más moderado. En cambio, en Demanda temporal y Rendimiento no hay evidencia suficiente, con esta muestra, de que la condición cambie la presión de tiempo percibida ni la autopercepción de rendimiento: como se discute a continuación,

eso no es una debilidad del orquestador, sino una pista sobre dónde está realmente su ventaja.

5.10.6 Discusión

Los resultados respaldan la hipótesis central de esta investigación: el índice RTLX global cae de 53.02 a 26.35 al pasar del flujo manual al orquestado, con una reducción grande y estadísticamente significativa en cinco de las seis subescalas (Demanda Mental, Demanda Física, Esfuerzo, Frustración y RTLX global).

La Condición B presenta mayor variabilidad que la Condición A. Esto se explica por dos sesiones con falla de *timeout* durante el ensamblaje, en las que la Frustración y la percepción de Rendimiento se deterioraron notablemente; el resto de las sesiones de la Condición B no muestra este patrón, por lo que la variabilidad se atribuye a fallas puntuales de infraestructura y no a una inestabilidad general del orquestador.

La Demanda Temporal es la única subescala que no mejora con el orquestador (68.75 frente a 63.75 en A), consistente con el tiempo de inferencia de Qwen 2.5 14B (Sección 5.9.5). Sin embargo, el Esfuerzo percibido en la Condición B es el más bajo de todo el estudio (18.75), lo que indica que los participantes prefieren un tiempo de espera mayor a tener que coordinar la consulta manualmente.

El hallazgo central de este análisis es que el orquestador no mejora la calidad de las respuestas de los asistentes especialistas: estas son equivalentes en ambas condiciones y los fallos de la Condición B se deben a infraestructura de red o a la paciencia del usuario, no a errores de los asistentes. Lo que el orquestador elimina es el trabajo de coordinación que en el flujo manual recae en el estudiante —identificar el asistente correcto, enviarle su fragmento, esperar la respuesta y ensamblar el resultado final—. Esta es la contribución central de la tesis: el problema de fragmentación descrito en la Sección 1.3 no era, ante todo, un problema de corrección de las respuestas, sino del esfuerzo mental necesario para coordinarlas y es en ese aspecto donde el modelo de orquestación propuesto muestra una mejora consistente y medible.

Capítulo 6

Conclusiones y trabajo futuro

6.1 Conclusiones generales

El problema que motivó esta investigación (Sección 1.3) era la fragmentación de los asistentes virtuales educativos: sistemas aislados, sin mecanismos estandarizados de comunicación, que incrementan la carga cognitiva del estudiante al obligarlo a coordinar manualmente herramientas que no se comunican entre sí. Para atender este problema se diseñó, implementó y evaluó un modelo de interoperabilidad basado en el protocolo abierto OFP y en el patrón de diseño estructural Adaptador (Capítulo 3), capaz de orquestar la colaboración entre asistentes virtuales heterogéneos preservando la soberanía de los datos de las instituciones educativas.

La hipótesis de esta tesis (Sección 1.4) planteaba que dicho modelo facilitaría una evaluación colaborativa más efectiva y produciría una mejora significativa en la calidad del aprendizaje personalizado y la eficiencia pedagógica, en comparación con sistemas aislados. La evidencia del Capítulo 5 permite confirmar esta hipótesis de forma parcial, y precisar en qué dimensión se cumple:

- No se confirma en la dimensión de corrección de la respuesta. La tasa de acierto de la tarea fue similar entre el flujo manual y el flujo orquestado (13/16 en el flujo manual y 12/16 en el flujo orquestado, Tabla 5.19): el modelo propuesto no demostró una mejora en la calidad de la respuesta final respecto al uso aislado y manual de los mismos asistentes especialistas.
- Sí se confirma en la dimensión de carga cognitiva. El modelo produjo una reducción estadísticamente significativa y de tamaño de efecto grande en la demanda mental, el esfuerzo y la frustración percibidos por el usuario, con el índice RTLX global cayendo de 53.02 a 26.35 (Sección 5.10.3).

En otras palabras, la contribución central del modelo de interoperabilidad propuesto no es hacer que el estudiante obtenga una mejor respuesta, sino reducir sustancialmente el costo cognitivo que le implica obtenerla, trasladando la carga de la coordinación logística del usuario hacia el sistema. Dado que el problema descrito en la Sección 1.3 era, ante todo, un problema de carga de trabajo mental derivado del aislamiento entre sistemas —y no un problema de que los asistentes especialistas fueran individualmente deficientes, como confirma la Fase 1—, esta distinción no

debilita la hipótesis original, sino que la sitúa con mayor precisión: la interoperabilidad propuesta resuelve la dimensión de fragmentación y carga cognitiva, mientras que la calidad intrínseca de cada respuesta permanece acotada por la capacidad individual de cada asistente especialista, algo que queda fuera del alcance de una capa de orquestación.

Un segundo hallazgo relevante es que el desempeño del sistema orquestado no está determinado por la capacidad individual de los asistentes especialistas —que la Fase 1 mostró alta y estable independientemente del tamaño del modelo—, sino por la capa de coordinación misma: la segmentación de la consulta, el ruteo semántico y, sobre todo, el ensamblaje final de respuestas concurrentes (Secciones 5.9.1 y 5.9.4). Esto reposiciona el foco de futuras optimizaciones de sistemas similares: invertir en asistentes especialistas más grandes aporta poco si el cuello de botella real está en la etapa de integración.

6.2 Cumplimiento de los objetivos

A continuación se revisa el grado de cumplimiento de cada uno de los objetivos específicos planteados en la Sección 1.5. Los seis objetivos se cumplieron en su totalidad, lo que permite considerar satisfecho el objetivo general de desarrollar un modelo que facilite la interoperabilidad de asistentes virtuales heterogéneos.

- Identificar los requerimientos de interoperabilidad entre asistentes virtuales educativos heterogéneos, analizando sus protocolos y limitaciones de comunicación. Este objetivo se cumplió mediante la revisión del estado del arte (Capítulo 2) y la identificación explícita de la brecha de investigación (Sección 2.5.3): la ausencia de un modelo agnóstico capaz de orquestar asistentes heterogéneos sin depender de servicios propietarios en la nube.
- Diseñar un modelo para estandarizar el intercambio de datos y la delegación de turnos entre asistentes virtuales. Este objetivo se cumplió a través de la especificación del modelo de interoperabilidad (Capítulo 3), materializado en la estructura de mensaje "sobre" bajo el protocolo OFP y en el patrón de diseño Adaptador para la traducción entre protocolos heterogéneos.
- Desarrollar un orquestador que coordine la colaboración entre asistentes virtuales heterogéneos, manejando descubrimiento, delegación y coherencia conversacional. Este objetivo se cumplió mediante la implementación del orquestador con clasificación semántica basada en *prompt engineering*, estrategias de ejecución paralela y secuencial, y el protocolo de registro dinámico en caliente que resuelve el descubrimiento de nuevos asistentes (Secciones 4.4 y 4.7.1).
- Implementar un prototipo del modelo, integrando varios asistentes virtuales educativos heterogéneos. Este objetivo se cumplió a través de la construcción de un ecosistema funcional de diez asistentes especializados sobre una infraestructura de contenedores, incluyendo adaptadores individuales y una arquitectura RAG para memoria de largo plazo (Capítulo 4).
- Definir e integrar un módulo de métricas para evaluar el desempeño individual y en conjunto de los asistentes virtuales. Este objetivo se cumplió mediante

la instrumentación de telemetría nativa del orquestador y de interfaces de evaluación individual basadas en similitud de coseno y F1-Score (Sección 4.8), que sostuvieron tanto la Fase 1 como la Fase 2 de la evaluación.
- Evaluar la interoperabilidad, el mantenimiento del contexto educativo y la carga cognitiva percibida por el usuario final, mediante el instrumento estandarizado *NASA-TLX*. Este objetivo se cumplió mediante las dos fases de evaluación técnica (Secciones 5.3 a 5.9) y la evaluación de carga cognitiva con el cuestionario RTLX (Sección 5.10), con un diseño de medidas repetidas contrabalanceado (n = 16).

6.3 Aportaciones de la investigación

Las principales aportaciones de esta investigación, derivadas del diseño del sistema y de su evaluación empírica, se resumen a continuación:

- Un modelo de interoperabilidad aplicado al dominio educativo, que traduce la brecha arquitectónica identificada en la Sección 2.5.3 —la ausencia de un mecanismo estandarizado y agnóstico a la nube para la delegación de tareas entre asistentes virtuales— en una arquitectura concreta basada en OFP y en el patrón Adaptador (Capítulo 3).
- Un mecanismo de registro dinámico en caliente que permite la extensibilidad del ecosistema bajo el principio de abierto/cerrado (SOLID), incorporando nuevos asistentes especializados sin interrumpir el servicio ni modificar el núcleo del orquestador (Sección 4.7.1).
- Un módulo de telemetría y evaluación integrado nativamente en la arquitectura, que permite auditar, de forma automatizada, tanto el desempeño aislado de cada asistente como las decisiones de ruteo del sistema completo (Sección 4.8).
- Evidencia empírica original sobre la ubicación del cuello de botella en sistemas multi-asistente educativos: la degradación de desempeño al integrar asistentes bajo un orquestador se origina en la fase de ensamblaje y no en la calidad individual de los especialistas (Sección 5.9.1) y la capacidad de enrutamiento —no la de respuesta especializada— es la habilidad que se beneficia de la escala del modelo (Sección 5.9.3).
- Una metodología de evaluación de carga cognitiva (diseño de medidas repetidas contrabalanceado, con cuestionario RTLX) adaptada específicamente para comparar un flujo manual de coordinación entre asistente virtual frente a un flujo orquestado, reutilizable para evaluar otros sistemas de interoperabilidad educativa (Sección 5.10).
- Una distinción conceptual relevante para el diseño de sistemas de interoperabilidad: el valor de una capa de orquestación puede residir en la reducción del costo cognitivo de coordinación, de forma independiente —y no necesariamente acompañada— de una mejora en la exactitud de la respuesta final (Sección 6.1).

6.4 Limitaciones del modelo

A pesar de los aportes descritos, el modelo de interoperabilidad propuesto y su proceso de evaluación presentan las siguientes limitaciones, que delimitan el alcance de las conclusiones obtenidas y constituyen líneas de trabajo futuro:

- Cuello de botella estructural en la fase de ensamblaje. Incluso con el modelo orquestador de mejor desempeño global (Qwen 2.5 14B), persisten fallos de ensamblaje en consultas híbridas que requieren consolidar respuestas extensas de tres o cuatro asistentes concurrentes (e.g., las preguntas H06 y H11, Tabla 5.17), atribuibles a desbordamiento de la ventana de contexto, *timeouts* o corrupción del formato OFP.
- Latencia elevada del modelo orquestador con mejor desempeño. Qwen 2.5 14B, la arquitectura más robusta identificada en la Sección 5.9.5, presenta un tiempo de inferencia superior a siete minutos por consulta híbrida, lo que limita su viabilidad en implementaciones que requieran respuesta en tiempo real.
- Calibración fija del umbral de clasificación semántica. El umbral de 0.75 usado para binarizar la similitud de coseno no resulta igualmente discriminante entre dominios (Sección 5.9.2): penaliza como fallo total respuestas semánticamente cercanas en dominios de prosa extensa (e.g., Historia) y clasifica como acierto respuestas con baja similitud continua en dominios de expresión breve (e.g., Matemáticas).
- Ausencia de una interfaz gráfica propia del orquestador. La evaluación de carga cognitiva (Sección 5.10.1) tuvo que administrarse mediante un cliente HTTP, lo que limita la posibilidad de generalizar los resultados a un escenario de uso real con una interfaz gráfica de producción.
- Alcance muestral limitado en la evaluación de carga cognitiva. La evaluación RTLX se realizó con dieciséis participantes, dos preguntas híbridas y un único modelo orquestador (Qwen 2.5 14B), lo que restringe la generalización estadística de los resultados a otras configuraciones del sistema.
- Modelos LLM evaluados y acotados a arquitecturas locales de 8 a 14 mil millones de parámetros. No se evaluaron modelos comerciales de frontera (e.g., GPT-4o, Gemini o Claude) como cerebro del orquestador, por lo que no puede garantizarse que los patrones observados —en particular el cuello de botella de ensamblaje— se repliquen o se atenúen en modelos de mayor escala o de inferencia en la nube.
- Ecosistema evaluado con un número acotado de asistentes especializados. Si bien el mecanismo de registro dinámico en caliente fue diseñado para escalar horizontalmente, su comportamiento con decenas de asistentes federados concurrentes no fue puesto a prueba empíricamente en esta investigación.
- El modelo no demuestra una mejora en la calidad de la respuesta final. Como se discutió en la Sección 6.1, la tasa de acierto de la tarea fue similar entre el flujo manual y el orquestado (13/16 vs. 12/16); el aporte del modelo se limita, dentro del alcance de esta evaluación, a la dimensión de carga cognitiva y no debe interpretarse como una mejora comprobada en la efectividad pedagógica.

6.5 Trabajo futuro

Las limitaciones descritas abren, a su vez, un conjunto de líneas de investigación que podrían fortalecer o extender el modelo propuesto. A continuación se enumeran las principales direcciones identificadas:

- Rediseñar la fase de ensamblaje de respuestas —e.g., mediante ensamblaje incremental por flujo *streaming* o consolidación jerárquica por bloques— para mitigar el cuello de botella estructural identificado en la Sección 5.9.4.
- Explorar mecanismos de calibración dinámica del umbral de clasificación semántica, ajustado por dominio de conocimiento, en lugar de un valor fijo global de 0.75.
- Desarrollar una interfaz gráfica de usuario nativa del orquestador que permita repetir la evaluación de carga cognitiva en un entorno más representativo del uso real.
- Ampliar la evaluación de carga cognitiva a una muestra mayor y más diversa de participantes, con un número mayor de consultas híbridas y distintos perfiles de usuario (e.g., estudiantes de distintos niveles educativos).
- Evaluar modelos orquestadores adicionales, incluyendo modelos comerciales alojados en la nube, para contrastar sus patrones de ruteo y ensamblaje con los obtenidos en modelos locales de 8 a 14 mil millones de parámetros.
- Incorporar un mecanismo de aprendizaje continuo offline para el clasificador de intenciones (e.g., mediante fine-tuning eficiente por LoRA sobre los datos de telemetría de ruteo, Sección 4.8), manteniendo el orquestador sin estado durante la inferencia mientras se habilita su mejora incremental entre versiones.
- Poner a prueba la escalabilidad horizontal real del mecanismo de registro dinámico en caliente con un número significativamente mayor de asistentes federados operando de forma concurrente.
- Explorar métricas semánticas complementarias a la similitud de coseno —e.g., evaluación mediante un modelo de lenguaje como juez (*LLM-as-judge*)— para mitigar las limitaciones del umbral fijo identificadas en la Sección 5.9.2.
- Diseñar estudios longitudinales que midan el efecto del modelo sobre indicadores reales de aprendizaje —retención de conocimiento y desempeño académico— más allá de la carga cognitiva percibida en una única sesión de uso.
- Fortalecer los mecanismos de seguridad y gobernanza de datos del modelo (autenticación entre nodos, políticas de privacidad diferenciadas por institución) de cara a un eventual despliegue multi-institucional en un entorno educativo real.

Anexo A

Estructura del repositorio de código

A continuación se detalla la organización jerárquica y la distribución de módulos dentro del repositorio de control de versiones del ecosistema distribuido. Esta estructura refleja la separación física entre el núcleo de orquestación, los esquemas de validación del protocolo OFP, la persistencia vectorial y los asistentes periféricos individuales.

```
.
├── README.md
├── docker-compose.yml
├── orchestrator
│   ├── Dockerfile
│   ├── metrics/
│   ├── orchestrator.py
│   └── requirements.txt
├── schemas
│   └── conversation-envelope-schema_v1.1.0.json
├── qdrant_storage
│   ├── aliases/
│   └── collections/
└── assistants
    ├── algebra                 <-- Estructura patrón de un asistente
    │   ├── Dockerfile
    │   ├── algebra1.pdf        <-- Corpus documental (RAG)
    │   ├── api_wrapper.py      <-- Adaptador del protocolo OFP
    │   └── bot_algebra.py      <-- Lógica de inferencia local
    ├── computacion/
    ├── espanol/
    ├── historia/
    ├── mathbuddy_avanzado/
    ├── mathbuddy_basico/
    ├── physics/
```

```
├── programacion/
├── quimica/
└── translator/
```

Declaración de dependencias del ecosistema

En este anexo se listan las bibliotecas y paquetes de software requeridos para la ejecución balanceada del módulo central de enrutamiento. El siguiente bloque de texto corresponde, de manera íntegra, al contenido del archivo `requirements.txt` del componente `orchestrator`:

```
fastapi
uvicorn
jsonschema
flask
requests
ollama
colorama
pandas
numpy
```

Anexo B

Manifiestos de contenerización (*Dockerfiles*)

Para garantizar la reproducibilidad, portabilidad e inmutabilidad del entorno de ejecución, se definió un archivo *Dockerfile* dedicado para cada microservicio. A modo de ejemplo representativo del patrón utilizado en el ecosistema, se expone a continuación el *Dockerfile* empleado para la construcción de la imagen del orquestador:

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir --upgrade pip && \
    pip install --no-cache-dir --default-timeout=1000 -r
       requirements.txt && \
    pip install --no-cache-dir flask requests ollama
COPY . .
EXPOSE 5000
CMD ["python", "orchestrator.py"]
```

Archivo `docker-compose.yml`

En este anexo se detalla la configuración declarativa de la infraestructura del ecosistema. Por motivos de concisión, se expone el núcleo de la red (motor de inferencia local, orquestador principal y motor vectorial) junto con un único asistente de dominio (Computación), el cual sirve como representación del patrón de diseño textitBackend-Wrapper que se replica homogéneamente para el resto de las disciplinas operativas.

```
# docker-compose.yml (Fragmento representativo)

services:
  ollama:
    image: ollama/ollama:latest
    container_name: ollama_server
    ports:
```

```
    - "11434:11434"
  volumes:
    - ollama_models:/root/.ollama
  restart: unless-stopped
  networks:
    - ofp_network
  environment:
    - OLLAMA_NUM_GPU_LAYERS=20     # Límite de carga en VRAM
    - OLLAMA_KEEP_ALIVE=-1
    - OLLAMA_MAX_LOADED_MODELS=1
    - OLLAMA_NUM_PARALLEL=1
    - OLLAMA_MAX_QUEUE=2
    - OLLAMA_NUM_THREADS=0

orchestrator_ollama:
  build: ./orchestrator
  container_name: orchestrator_ollama
  ports:
    - "5000:5000"
  environment:
    - OLLAMA_HOST=http://ollama:11434
  depends_on:
    - ollama
  volumes:
    - ./schemas:/app/schemas
    - ./orchestrator/metrics:/app/metrics
  networks:
    - ofp_network

qdrant:
  image: qdrant/qdrant:latest
  container_name: qdrant_db
  ports:
    - "6333:6333"
  volumes:
    - ./qdrant_storage:/qdrant/storage # Persistencia vectorial
       local
  networks:
    - ofp_network

# ====================== ASISTENTE REPRESENTATIVO (COMPUTACIÓN)
   ======================
computacion-backend:
  build: ./assistants/computacion
  container_name: computacion-backend
  command: python bot_computacion.py
  environment:
    - OLLAMA_HOST=http://ollama:11434
  depends_on:
    - ollama
    - qdrant
  networks:
    - ofp_network
  restart: unless-stopped
```

```
  assistant_computacion:
    build: ./assistants/computacion
    container_name: assistant_computacion
    ports:
      - "5014:5014"
    command: python api_wrapper.py
    environment:
      - COMP_INTERNAL_URL=http://computacion-backend:5000/preguntar
    depends_on:
      - computacion-backend
    volumes:
      - ./schemas:/app/schemas:ro
    networks:
      - ofp_network
    restart: unless-stopped

  # ... [Configuración de los asistentes de Física, Matemáticas,
  #      Historia, Traducción, Química, etc. omitida por brevedad.
  #      Todos replican estrictamente el patrón topológico
     superior] ...

networks:
  ofp_network:
    driver: bridge

volumes:
  ollama_models:
```

Anexo C

Esquemas y cargas útiles del protocolo OFP

En este anexo se documentan las estructuras de datos estandarizadas que habilitan la comunicación basada en eventos dentro de la red interna del modelo.

Estructura base del sobre

El siguiente bloque en formato JSON representa la envoltura inmutable que transporta los metadatos de enrutamiento, el identificador de sesión (UUID v4) y la firma del emisor, encapsulando los eventos dentro del objeto raíz `openFloor`:

```
{
  "openFloor": {
    "schema": {
      "version": "1.1.0"
    },
    "conversation": {
      "id": "f47ac10b-58cc-4372-a567-0e02b2c3d479"
    },
    "sender": {
      "speakerUri": "tag:orchestrator.local,2026:orch001"
    },
    "events": [ ... ]
  }
}
```

Evento de emisión *(utterance)*

A continuación se ilustra la estructura detallada del evento *utterance* introducido en el arreglo de eventos. Este formato define el marco temporal de la interacción e inyecta la carga útil de texto natural mediante el esquema de *tokens*:

```
{
```

```
  "eventType": "utterance",
  "parameters": {
    "dialogEvent": {
      "speakerUri": "tag:user.local,2026:user001",
      "span": {
        "startTime": "2026-07-17T17:33:20.000Z",
        "endTime": "2026-07-17T17:33:20.000Z"
      },
      "features": {
        "text": {
          "mimeType": "text/plain",
          "tokens": [
            { "value": "Explica la diferencia entre la
               memoria RAM y la memoria ROM." }
          ]
        }
      }
    }
  }
}
```

Esquema de validación

El siguiente fragmento corresponde al contrato `conversation-envelope-schema_v1.1.0.json`, el cual emplea la especificación JSON Schema para definir las restricciones estructurales obligatorias que previenen el procesamiento de paquetes malformados en el orquestador:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "OFP Envelope Validation Schema",
  "type": "object",
  "required": ["openFloor"],
  "properties": {
    "openFloor": {
      "type": "object",
      "required": ["schema", "conversation", "events"],
      "properties": {
        "schema": {
          "type": "object",
          "required": ["version"],
          "properties": {
            "version": {
              "type": "string",
              "pattern": "^[0-9]+\\.[0-9]+\\.[0-9]+$"
            }
          }
```

```
        },
        "conversation": {
          "type": "object",
          "required": ["id"],
          "properties": {
            "id": {
              "type": "string",
              "format": "uuid"
            }
          }
        },
        "events": {
          "type": "array",
          "items": {
            "type": "object",
            "required": ["eventType", "parameters"]
          }
        }
      }
    }
  }
}
```

Anexo D

Lógica de orquestación y plantillas de *prompt engineering*

En este anexo se documenta el código fuente y las plantillas de *prompt engineering* que conforman el núcleo lógico del orquestador. Estas estructuras facilitan la clasificación semántica, la exclusión mutua de tareas, la concurrencia mediante hilos y la propagación del contexto entre asistentes.

Plantilla de clasificación y enrutamiento semántico

La clasificación de las consultas se implementa mediante un *prompt* determinista que instruye al modelo LLM para analizar la intención del usuario. Esta directriz obliga al modelo a segmentar la consulta, asignar cada parte al dominio experto correspondiente y planificar el orden de ejecución. Para garantizar este comportamiento determinista —es decir, que el modelo produzca siempre la misma salida estructurada ante una misma consulta y no alucine campos adicionales en el JSON—, la invocación a la API se realiza configurando el hiperparámetro de temperatura estrictamente en cero (`temperature: 0.0`). A continuación se expone la plantilla y la ejecución exacta utilizada en el núcleo del orquestador:

```
def clasificar_con_llm_estandarizado(user_query):
    prompt = f"""Eres el enrutador central de un framework de
       asistentes educativos.Tu tarea es leer la consulta del
       usuario, dividirla lógicamente, asignar el asistentes
       correcto a cada segmento y determinar el ORDEN de
       ejecución.

Consulta del usuario:
"{user_query}"

Asistentes disponibles:
- fisica: leyes de la física, cinemática, dinámica, energía,
   fórmulas físicas (ej. W = F * d).
```

- mathbuddy_basico: aritmética (suma, resta, multiplicación, división, fracciones), jerarquía de operaciones (PEMDAS), ecuaciones de primer grado.
- mathbuddy_avanzado: cálculo (derivadas, integrales, límites), factorización de límites indeterminados.
- algebra: polinomios, factorización, diferencia de cuadrados, signos de agrupación, ecuaciones algebraicas.
- computacion: ESTRICTAMENTE hardware, componentes físicos (RAM, ROM, CPU, discos duros), redes y sistemas operativos. PROHIBIDO enviar conceptos de código o estructuras de datos aquí.
- programacion: ESTRICTAMENTE desarrollo de software, código, lenguajes (C++), memoria dinámica, estructuras de datos (arreglos bidimensionales), paso por valor/referencia, y paradigmas (herencia, polimorfismo, public/private).
- espanol: ESTRICTAMENTE ortografía, gramática, sintaxis, tildes, y signos de puntuación. PROHIBIDO enviar definiciones de palabras técnicas de otras materias (ej. no envíes "arreglo bidimensional" ni "enlace covalente" a español).
- historia: eventos históricos, guerras, imperios, biografías, modelos de gobierno (colonialismo, imperialismo, fascismo).
- quimica: átomos, isótopos, enlaces (iónico/covalente), configuración electrónica, moles, Ley de Conservación de la Masa.
- translator: solicitudes de traducción a cualquier otro idioma.

REGLAS VITALES DE RUTEO:
1. Divide la consulta en segmentos y asigna un asistente. PROHIBIDO OMITIR INFORMACIÓN. Cada acción, pregunta o tema solicitado por el usuario DEBE estar asignado a un asistente en los segmentos. No dejes ninguna parte de la consulta original sin atender.
2. Determina las ‘‘etapas’’ de ejecución en un arreglo de arreglos.
3. REGLA ESTRICTA DE TRADUCCIÓN (EXCLUSIÓN MUTUA):
   - CASO A (Traducción Global): Si el usuario pide traducir TODA la respuesta combinada (ej. "traduce toda la respuesta al inglés"), DEBES marcar "requiere_traduccion": true e indicar el "idioma_objetivo". EN ESTE CASO, TIENES PROHIBIDO agregar al asistente "translator" en "etapas" o en "segmentos".
   - CASO B (Subtarea de Traducción): Si el usuario pide traducir solo una frase específica o pregunta cómo se dice algo (ej. "traduce la frase X", "cómo se dice

```
      perro en francés"), DEBES marcar "requiere_traduccion":
      false. EN ESTE CASO, SÍ debes agregar a "translator"
      como un asistente normal en las "etapas" y en los
      "segmentos".
4. Devuelve SOLO un JSON válido.

Formato esperado:
{{
  "etapas": [
    ["fisica", "historia"],
    ["mathbuddy_basico"]
  ],
  "segmentos": {{
    "fisica": "Dame la fórmula de la Segunda Ley de Newton",
    "historia": "Quién fue Newton",
    "mathbuddy_basico": "Calcula la fuerza si masa=10 y
       aceleracion=5 usando la fórmula"
  }},
  "requiere_traduccion": false,
  "idioma_objetivo": null
}}
"""
    try:
        response = ollama_client.chat(
            model=MODELO_CLASIFICACION,
            format='json',
            messages=[{'role': 'user', 'content': prompt}],
            options={"temperature": 0.0}  # Control
               determinista del LLM
        )
        # ... (Deserialización del JSON omitida por brevedad)
           ...
```

Ejecución paralela y propagación de contexto

La estrategia de ejecución paralela y secuencial descrita en el Capítulo 4 se implementa mediante la creación de hilos del sistema operativo (`threading.Thread`). Los asistentes virtuales que pertenecen a una misma etapa se invocan de forma concurrente (en paralelo), mientras que el comando de sincronización (`t.join()`) bloquea el avance hasta que todos los procesos de esa etapa concluyan.

Además, para manejar la dependencia de contexto en ejecuciones secuenciales, se implementó un mecanismo de inyección dinámica. Si un asistente requiere los cálculos o datos generados en una etapa anterior, se le transfiere el contexto acumulado junto con instrucciones estrictas para evitar que confunda su rol o duplique la información en la respuesta final, tal como se observa en la rutina `call_agent`:

```
def call_agent(asistente):
```

```
    texto_base = segmentos.get(asistente, "")
    if not texto_base: return

    # --- 1. AISLAMIENTO Y MANEJO DE CONTEXTO ---
    if asistente == "translator":
        texto_a_enviar = f"TRADUCE ÚNICAMENTE ESTA SOLICITUD.
           NO TRADUZCAS NI REPITAS NADA MÁS: {texto_base}"
    else:
        if idx_etapa > 0 and contexto_acumulado:
            texto_a_enviar = f"{texto_base}\n\n[ATENCIÓN:
               Aquí tienes respuestas de otros asistentes. Ú
               salas SOLO si necesitas sus fórmulas o datos
               para tu tarea. Tienes PROHIBIDO repetir o
               copiar esta información en tu respuesta
               final]:\n{contexto_acumulado}"
        else:
            texto_a_enviar = texto_base

    # ... (Llamada HTTP y manejo del sobre OFP) ...
```

Fase de síntesis e inyección de formato LaTeX

El siguiente bloque lógico ilustra cómo el orquestador agrupa las respuestas crudas e inyecta directrices de homogeneización matemática, preparando la salida para su renderizado en interfaces *frontend*:

```
def construir_prompt_sintesis(respuestas_asistentes):
    contexto_acumulado = "\n\n".join(
        [f"Agente {r['asistente']}: {r['texto']}" for r in
           respuestas_asistentes]
    )
    prompt_sintesis = f"""
    Eres el asistente unificador del sistema. A continuación
       se presentan las respuestas de varios asistentes
       expertos.
    Respuestas:
    {contexto_acumulado}

    Reglas de formato:
    1. Redacta una respuesta cohesiva y fluida que integre
       toda la información de los expertos.
    2. No contradigas los datos técnicos provistos.
    3. Homogeneización matemática: Toda ecuación o fórmula
       DEBE estar encerrada en símbolos de dólar dobles ($$)
       para su correcto renderizado en KaTeX.
    """
    return prompt_sintesis
```

Plantilla de traducción global

En lugar de recurrir a una fase de síntesis computacionalmente costosa, el orquestador ensambla de manera directa los fragmentos de conocimiento mediante el empaquetado OFP (`create_collaborative_response`). La única reescritura que ocurre a nivel del orquestador se da cuando el clasificador activa la bandera `requiere_traduccion` (Regla de Exclusión Mutua A). En dicho escenario, se consolida el texto final y se ejecuta una segunda instrucción de *Prompt Engineering* dedicada exclusivamente a la traducción:

```
if requiere_traduccion and "translator" in AGENTS:
    texto_completo = "\n".join([info.get("text", "") for ag,
       info in responses_map.items() if info.get("text") and
       ag != "translator"])

    if texto_completo.strip():
        idioma_final = idioma_objetivo if idioma_objetivo
           else "el idioma solicitado"
        prompt_trans = f"""Actúa como un traductor
           profesional. Tu única tarea es traducir TODO el
           texto que se encuentra entre las etiquetas <TEXTO>
           y </TEXTO> al {idioma_final}. Es obligatorio que
           mantengas todos los párrafos originales. No
           agregues notas, explicaciones ni saludos, devuelve
           únicamente la traducción.
<TEXTO>
{texto_completo}
</TEXTO>"""

        # ... (Invocación al asistente traductor) ...
```

Anexo E

Implementación de adaptadores y bases vectoriales

En este anexo se detalla el código fuente correspondiente al patrón Adaptador utilizado para garantizar la interoperabilidad del protocolo OFP, así como las rutinas de inicialización vectorial y las plantillas para RAG.

Capa de intermediación (API Wrapper)

El siguiente fragmento, implementado mediante el marco de trabajo Flask, ilustra cómo el componente contenedor (*wrapper*) intercepta el mensaje OFP, lo traduce para el motor interno y re-empaqueta la respuesta respetando el estándar de la estructura `openFloor`:

```
@app.route('/receive', methods=['POST'])
def handle_ofp_request():
    # 1. Recepcion de openFloor
    ofp = request.json.get("openFloor", {})
    conv_id = ofp.get("conversation", {}).get("id")

    # 2. Extraccion de texto
    eventos = ofp.get("events", [])
    user_text = ""
    if eventos and eventos[0].get("eventType") == "utterance":
        dg = eventos[0]["parameters"]["dialogEvent"]
        user_text =
           dg["features"]["text"]["tokens"][0]["value"]

    # 3. Llamada al backend interno
    bot_ans =
       requests.post(os.environ.get("INTERNAL_BACKEND_URL"),
              json={"prompt":
                 user_text}).json().get("respuesta", "")
```

```
    # 4. Construccion de respuesta OFP
    response = {
        "openFloor": {
            "schema": {"version": "1.1.0"},
            "conversation": {"id": conv_id},
            "events": [{
                "eventType": "utterance",
                "parameters": {"dialogEvent": {"features":
                   {"text": {
                    "mimeType": "text/plain",
                    "tokens": [{"value": bot_ans}]
                }}}}
            }]
        }
    }
    return jsonify(response), 200
```

Inicialización del espacio vectorial (Qdrant)

El siguiente bloque lógico gestiona la preexistencia y creación de colecciones en la base de datos vectorial Qdrant, configurando la métrica de Distancia del Coseno para la evaluación de similitud semántica:

```
from qdrant_client import QdrantClient
from qdrant_client.models import Distance, VectorParams

# Conexión al contenedor de Qdrant en la red interna
client = QdrantClient(host="qdrant_db", port=6333)

def inicializar_coleccion(nombre_coleccion, dimension_vector):
  # Verificación de preexistencia para evitar re-ingesta
     costosa
  if not
     client.collection_exists(collection_name=nombre_coleccion):
     client.create_collection(
            collection_name=nombre_coleccion,
            vectors_config=VectorParams(
                size=dimension_vector,
                distance=Distance.COSINE
            ),
        )
        print(f"Colección {nombre_coleccion} inicializada
           exitosamente.")
```

Plantilla para asistentes virtuales basados en RAG

Para limitar la propensión a la alucinación en los modelos generativos, se emplea la siguiente plantilla que inyecta dinámicamente los fragmentos recuperados de la literatura académica:

```
def generar_prompt_rag(consulta_usuario,
   fragmentos_recuperados):
    contexto_unificado = "\n".join(fragmentos_recuperados)

    prompt = f"""
    Eres un tutor académico experto. Utiliza ÚNICAMENTE la
       siguiente información de referencia para responder a
       la pregunta del estudiante. Si la respuesta no se
       encuentra en la referencia, indica explícitamente que
       no posees la información. No inventes datos.

    --- REFERENCIA ACADÉMICA ---
    {contexto_unificado}
    ------------------------------
    Pregunta del estudiante: {consulta_usuario}
    Respuesta:
    """
    return prompt
```

Anexo F

Especificación de la interfaz de consumo (API REST)

En este anexo se documenta la estructura de los objetos de transferencia de datos (*Data Transfer Objects*) utilizados en la capa de presentación del modelo, ilustrando la entrada de datos y la salida homogeneizada bajo el protocolo OFP.

Estructura de la petición HTTP (entrada)

El siguiente bloque JSON ejemplifica una solicitud dirigida al orquestador mediante el método HTTP POST, planteando un problema que requiere la colaboración de los dominios de Física, Matemática Avanzada y traducción:

```
POST /query
Content-Type: application/json

{
  "query": "Un cohete tiene una masa de 1200 kg y acelera a
     15 m/s^2. Calcula la fuerza necesaria usando la fórmula
     F=m*a, luego obtén la derivada de esa fuerza respecto al
     tiempo si fuera constante y traduce todo el análisis al
     inglés."
}
```

Estructura de la respuesta colaborativa (salida)

A continuación se ilustra la respuesta final del sistema tras el proceso de enrutamiento, ejecución secuencial y traducción global. A diferencia de un JSON convencional, la API respeta estrictamente el contrato OFP, encapsulando los metadatos de la orquestación y el texto traducido dentro del arreglo de eventos. Nótese la estandarización tipográfica de las ecuaciones utilizando delimitadores de LaTeX (`$$`), lo cual permite su posterior renderizado en motores visuales como KaTeX:

```
{
  "openFloor": {
    "schema": { "version": "1.1.0" },
    "conversation": { "id":
       "f47ac10b-58cc-4372-a567-0e02b2c3d479" },
    "sender": { "speakerUri":
       "tag:orchestrator.local,2026:orch001" },
    "events": [
      {
        "eventType": "orchestration_metadata",
        "parameters": {
          "tipo_ejecucion": "secuencial",
          "total_etapas": 2
        }
      },
      {
        "eventType": "utterance",
        "parameters": {
          "dialogEvent": {
            "speakerUri":
               "tag:translator_global.local,2026:agent001",
            "span": {
              "startTime": "2026-07-17T18:35:00.000Z",
              "endTime": "2026-07-17T18:35:05.320Z"
            },
            "features": {
              "text": {
                "mimeType": "text/plain",
                "tokens": [
                  {
                    "value": "The force is calculated
                       as:\n\n$$F = m \\cdot a = 1200 \\,
                       \\text{kg} \\cdot 15 \\, \\text{m/s}^2
                       = 18000 \\, \\text{N}$$\n\nSince the
                       force is constant, its derivative with
                       respect to time
                       is:\n\n$$\\frac{dF}{dt} = 0$$"
                  }
                ]
              },
              "context": {
                "mimeType": "text/plain",
                "tokens": [{"value": "Traducción Global al
                   inglés"}]
              }
            }
          }
        }
```

```
      },
      {
        "eventType": "yieldFloor",
        "parameters": {}
      }
    ]
  }
}
```

Anexo G

Manifiestos de registro dinámico en caliente

En este anexo se ilustra la especificación estructural del evento de registro empleado por los componentes externos para unirse a la colaboración entre asistentes virtuales educativos. El siguiente bloque JSON representa el manifiesto emitido de forma automatizada por el adaptador de un nuevo servicio al finalizar su secuencia de inicialización, encapsulado estrictamente bajo el estándar del protocolo OFP:

```
{
  "openFloor": {
    "schema": { "version": "1.1.0" },
    "conversation": { "id": "system-registration-uuid" },
    "sender": { "speakerUri":
       "tag:assistant_biologia.local,2026:agent001" },
    "events": [
      {
        "eventType": "publishManifest",
        "parameters": {
          "manifest": {
            "agent_id": "biologia",
            "endpoint":
               "http://assistant_biologia:5019/receive",
            "capabilities": [
              "biología celular",
              "genética",
              "anatomía"
            ],
            "description": "Experto en ciencias biológicas
               basado en RAG."
          }}
        } ]
      }}
```

Anexo H

Scripts de instrumentación y trazabilidad

En este anexo se documentan las rutinas lógicas implementadas para habilitar la trazabilidad del ecosistema y la evaluación automatizada del modelo.

Rutina de vectorización automatizada

El siguiente fragmento ilustra cómo el módulo de pruebas interactúa con el modelo de incrustaciones (`mxbai-embed-large`) para proyectar los textos en un espacio vectorial latente, permitiendo cálculos algorítmicos posteriores de similitud semántica:

```
def obtener_embedding(texto, max_reintentos=3):
    for intento in range(max_reintentos):
        try:
            resp = requests.post(
                OLLAMA_URL,
                json={"model": MODELO_EMBEDDING, "prompt":
                   texto},
                timeout=30
            )
            resp.raise_for_status()
            # Retorna el vector formateado para operaciones
               con NumPy
            return
               np.array(resp.json()["embedding"]).reshape(1,
               -1)
        except requests.exceptions.RequestException:
            time.sleep(3) # Pausa táctica ante saturación de
               red
    return None
```

Inyección de métricas y trazabilidad en el protocolo OFP

A continuación se detalla el bloque lógico ejecutado por el orquestador para inyectar métricas de latencia (`span`) y trazabilidad de delegación (`context`) dentro de la respuesta colaborativa final:

```
# Cálculo de la ventana temporal
now = datetime.utcnow()
start_iso = now.isoformat() + "Z"
end_iso = (now + timedelta(seconds=latencia)).isoformat() +
   "Z"

# Inyección de metadatos en el historial de eventos OFP
events.append({
    "eventType": "utterance",
    "parameters": {
        "dialogEvent": {
            "speakerUri":
               f"tag:{asistente_id}.local,2026:agent001",
            "span": {"startTime": start_iso, "endTime":
               end_iso},
            "features": {
                "text": {"mimeType": "text/plain", "tokens":
                   [{"value": texto_generado}]},
                "context": {"mimeType": "text/plain",
                   "tokens": [{"value": fragmento_consulta}]}
            }
        }
    }
})
```

Gestor de ciclo de vida de memoria VRAM

Para proveer tolerancia a fallos durante iteraciones masivas de pruebas, el siguiente *script* fuerza la liberación de tensores en la memoria gráfica enviando la directiva `keep_alive: 0`:

```
def limpiar_memoria_ollama():
    """Fuerza la descarga de los LLM de la memoria RAM/VRAM"""
    for modelo in MODELOS_A_LIMPIAR:
        try:
            requests.post(
                OLLAMA_GENERATE_URL,
                json={"model": modelo, "keep_alive": 0},
                timeout=5
            )
```

```
    except Exception:
        pass
# Tiempo de gracia para recolección de basura del sistema
   operativo
time.sleep(5)
```

Anexo I

Banco de consultas de evaluación

A continuación se presenta el listado de las 35 consultas utilizadas durante la fase de evaluación del sistema, detallando su identificador (**ID**) y el texto íntegro de la pregunta:

H01 Explica cómo fue el descubrimiento del radio por Marie Curie, indica cuál es el símbolo químico del radio en la tabla periódica y traduce toda la respuesta al inglés.

H02 ¿Cuál es la fórmula de la Segunda Ley de Newton? Usando esa fórmula, calcula la fuerza si la masa es 10 y la aceleración es 5, y luego traduce la explicación al francés.

H03 ¿Cuál es la diferencia entre la memoria RAM y ROM a nivel de hardware, cómo se maneja la memoria dinámica con punteros en C++ y cómo se corrige ortográficamente la frase 'la ram es mui rrapida'?

H04 ¿Qué factores permitieron a Estados Unidos fortalecer su hegemonía mundial tras la Primera Guerra Mundial, qué es la presión en Física y cuál es la derivada de la presión P respecto al volumen V si $P = T/V$?

H05 Explica brevemente la diferencia entre el verbo 'ser' y 'estar' en español, qué es un sistema operativo y sus objetivos, y traduce todo el resultado al italiano.

H06 ¿Cuál es la diferencia entre Aritmética y Álgebra, para qué sirven los operadores `new` y `delete` en C++, cuánto es 20/4 y traduce todo el resultado al portugués?

H07 Resume quién fue Albert Einstein, explica qué es la energía cinética, qué es un enlace covalente y cómo funciona el sistema binario basado en transistores.

H08 ¿Cuál es la regla general para la suma de polinomios, cuál es la derivada de $x^3 + 2x$, y qué afirma la Tercera Ley de Newton?

H09 Explica qué es un átomo y sus partículas, resuelve la ecuación $x + 5 = 12$, indica cuándo llevan tilde las palabras 'quién' y 'cuál', e integra la función $\sin(x)$.

H10 Dime en qué año inició la Revolución Mexicana, qué es un isótopo, qué es el polimorfismo en programación, calcula 8×7 y dime cómo se traduce 'Muchas gracias' en japonés.

H11 ¿Cuáles son los signos de agrupación en álgebra, qué es una variable de tipo puntero en C++, cuánto es $1/2 + 1/4$ y traduce todo al francés?

H12 ¿Qué es la placa base o tarjeta madre, qué significa la dimensión de una cantidad física, cuál es la derivada de $x \cdot \sin(x)$ usando la regla del producto y traduce todo al italiano?

H13 ¿En qué consiste el factor común en factorización, qué son los isótopos, cuál es la diferencia entre hardware y software, y cuál es la diferencia entre palabras agudas, graves y esdrújulas?

H14 ¿Cuáles fueron los objetivos de la ONU, cómo se define el concepto de mol, cuánto es $15 + 27$ y traduce todo al portugués?

H15 ¿Qué factores permitieron la hegemonía de Estados Unidos, cuál es el objetivo de un sistema operativo, y cuál es la derivada de $\log(x)$?

H16 ¿Qué función principal cumple el adjetivo calificativo, qué indican los modificadores `public`, `private` y `protected` en programación orientada a objetos y qué establece la Ley de Conservación de la Masa de Lavoisier?

H17 Explica cómo se maneja la memoria dinámica en C++, dame la fórmula del trabajo en Física ($W = F \cdot d$) y asume $F = 20$ y $d = 2$ para calcular el resultado matemático final en esta fórmula.

H18 ¿Qué aportó el judaísmo ideológicamente, cuál es la diferencia entre enlace iónico y covalente, y traduce todo al alemán?

H19 ¿Cuál es la diferencia entre Aritmética y Álgebra, qué es la CPU, cuál es la diferencia entre rapidez y velocidad, y simplifica la fracción $4/8$?

H20 ¿Qué estrategias políticas permitieron la hegemonía del Imperio Romano, qué es la electronegatividad, qué es un constructor en C++ y traduce todo a inglés?

H21 ¿Qué diferencias existen entre el colonialismo y el imperialismo, cuándo se usa el punto y coma, y cuál es la derivada de $\tan(x)$?

H22 ¿Por qué los imperios precolombinos fueron derrotados, qué plantea la Ley de Conservación de la Energía y cuál es la diferencia entre paso por valor y por referencia en programación?

H23 ¿Qué impacto tuvo la agricultura en la humanidad, cuál es la diferencia entre ácidos y bases según Arrhenius, calcula $5 + 3 \times 2$ y traduce todo a francés?

H24 ¿Qué innovación táctica implementó la Convención Francesa en 1793, cómo se escribe el prefijo ‘ex’ junto a ‘primer ministro’, cuál es la regla para diferencia de cuadrados perfectos y traduce todo el resultado a portugués?

H25 ¿Por qué Rusia no era ideal para la revolución según Lenin, qué es un arreglo bidimensional en programación y cuál es la integral de $\sin(x)$?

H26 ¿Cómo logró Mussolini el régimen fascista, de dónde provienen las letras s, p, d, f, cuál es el propósito del disco duro o SSD y traduce todo a inglés?

H27 Explica el concepto de velocidad instantánea, qué es la configuración electrónica de un átomo y resuelve la ecuación $3x = 15$.

H28 ¿Qué establece la Primera Ley de Newton, cuáles son las excepciones para omitir los signos de interrogación, qué es la herencia en C++ y traduce todo a alemán?

H29 ¿Cómo se define el trabajo en Física Mecánica, cómo se halla el mínimo común múltiplo de monomios y qué es la ALU?

H30 ¿Qué afirma la Tercera Ley de Newton, a qué se le llama software de aplicación, cuál es la derivada de $x^3 - 4x$ y traduce todo a italiano?

H31 ¿Qué es la energía potencial gravitatoria, la palabra ‘enlace’ es aguda o grave, qué es la transposición de términos y traduce todo a francés?

H32 ¿Qué es la presión y sus unidades, cuáles son las tres fases del proceso de programación y resuelve $x - 4 = 10$?

H33 ¿Qué fue la ENIAC, a qué es igual el cuadrado de la suma de dos cantidades, integra $x^3 - 2x$ y traduce todo a portugués?

H34 ¿Cuál es la diferencia entre por qué, porque y porqué, qué es la herencia, calcula 10 menos 2 entre 4 y traduce todo el resultado a inglés?

H35 ¿Para qué se usa `virtual` en C++, es obligatorio el uso de coma antes de la conjunción (y) en la gramática española, cuál es el límite de $(x^2 + 2x)/x$ en 0 y traduce todo el resultado a inglés?

Anexo J

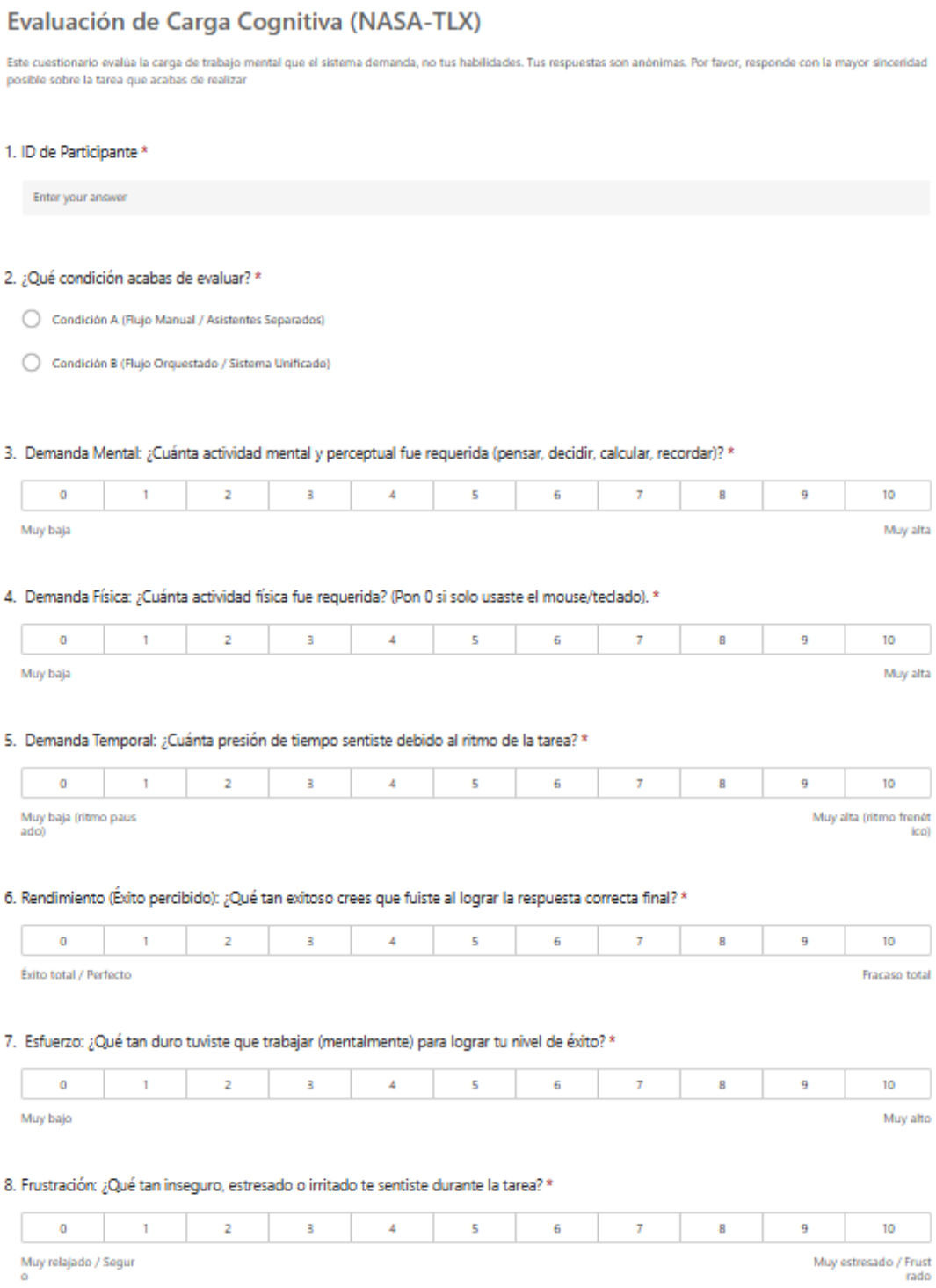
Instrumento de evaluación: cuestionario NASA-TLX (RTLX)

El presente instrumento, basado en la variante Raw TLX (RTLX), se administró digitalmente mediante la plataforma Microsoft Forms, a los participantes, inmediatamente después de concluir cada condición experimental. Se solicitó a los evaluadores marcar una puntuación en una escala lineal numérica del 0 al 10, cuyos resultados fueron posteriormente estandarizados a una escala porcentual (0-100) para el análisis estadístico.

Las dimensiones evaluadas, junto con sus anclajes semánticos en los extremos, fueron las siguientes:

1. **Demanda Mental:** ¿Cuánta actividad mental y perceptual fue requerida (pensar, decidir, calcular, recordar)? (0: Muy baja / 100: Muy alta).
2. **Demanda Física:** ¿Cuánta actividad física fue requerida? (0: Muy baja / 100: Muy alta).
3. **Demanda Temporal:** ¿Cuánta presión de tiempo sintió debido al ritmo al que ocurrieron las tareas? (0: Muy baja / 100: Muy alta).
4. **Rendimiento (*Performance*):** ¿Qué tan exitoso cree que fue al lograr los objetivos de la tarea? (0: Éxito total / 100: Fracaso total). Nota: Escala invertida para el cálculo de carga global.
5. **Esfuerzo:** ¿Qué tan duro tuvo que trabajar mentalmente para alcanzar su nivel de rendimiento? (0: Muy bajo / 100: Muy alto).
6. **Frustración:** ¿Qué tan inseguro, estresado o irritado se sintió durante la tarea? (0: Muy relajado / 100: Muy estresado).

A continuación, en la Figura J.1, se presenta una captura de pantalla del instrumento digital tal como fue desplegado a los participantes mediante la plataforma Microsoft Forms.



Evaluación de Carga Cognitiva (NASA-TLX)

Este cuestionario evalúa la carga de trabajo mental que el sistema demanda, no tus habilidades. Tus respuestas son anónimas. Por favor, responde con la mayor sinceridad posible sobre la tarea que acabas de realizar

1. ID de Participante *

Enter your answer

2. ¿Qué condición acabas de evaluar? *

- Condición A (Flujo Manual / Asistentes Separados)
- Condición B (Flujo Orquestado / Sistema Unificado)

3. Demanda Mental: ¿Cuánta actividad mental y perceptual fue requerida (pensar, decidir, calcular, recordar)? *

0	1	2	3	4	5	6	7	8	9	10

Muy baja — Muy alta

4. Demanda Física: ¿Cuánta actividad física fue requerida? (Pon 0 si solo usaste el mouse/teclado). *

0	1	2	3	4	5	6	7	8	9	10

Muy baja — Muy alta

5. Demanda Temporal: ¿Cuánta presión de tiempo sentiste debido al ritmo de la tarea? *

0	1	2	3	4	5	6	7	8	9	10

Muy baja (ritmo pausado) — Muy alta (ritmo frenético)

6. Rendimiento (Éxito percibido): ¿Qué tan exitoso crees que fuiste al lograr la respuesta correcta final? *

0	1	2	3	4	5	6	7	8	9	10

Éxito total / Perfecto — Fracaso total

7. Esfuerzo: ¿Qué tan duro tuviste que trabajar (mentalmente) para lograr tu nivel de éxito? *

0	1	2	3	4	5	6	7	8	9	10

Muy bajo — Muy alto

8. Frustración: ¿Qué tan inseguro, estresado o irritado te sentiste durante la tarea? *

0	1	2	3	4	5	6	7	8	9	10

Muy relajado / Seguro — Muy estresado / Frustrado

Figura J.1: Vista de la interfaz del cuestionario NASA-TLX administrado a los participantes.

Bibliografía

[1] Harry Barton Essel, Dimitrios Vlachopoulos, Akwasi Tachie-Menson, Esi Eduafua Johnson, and Papa Kwasi Baah. The Impact of a Virtual Teaching Assistant (Chatbot) on Students' Learning in Ghanaian Higher Education. *International Journal of Educational Technology in Higher Education*, 19(1):1–22, 2022. doi:10.1186/s41239-022-00362-6.

[2] Helen Crompton and Diane Burke. Artificial Intelligence in Higher Education: The State of the Field. *International Journal of Educational Technology in Higher Education*, 20(1):22, 2023. doi:10.1186/s41239-023-00392-8.

[3] Ed Seidewitz. What models mean. *IEEE Software*, 20(5):26–32, 2003. doi:10.1109/MS.2003.1231147.

[4] Ian Sommerville. *Software Engineering*. Pearson, 9th edition, 2011. ISBN: 978-0-13-703515-1.

[5] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18, 03 2024. doi:10.1007/s11704-024-40231-1.

[6] Organisation for Economic Co-operation and Development. Oecd digital education outlook 2023: Towards an effective digital education ecosystem. Technical report, OECD Publishing, Paris, 2023. URL: https://www.oecd-ilibrary.org/education/oecd-digital-education-outlook-2023_c74f03de-en, doi:10.1787/c74f03de-en.

[7] UNESCO. *AI and Education: Guidance for Policy-makers*. Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura, 2021. ISBN: 978-92-3-100447-6. doi:10.54675/PCSP7350.

[8] Wayne Holmes, Kaska Porayska-Pomsta, Ken Holstein, Emma Sutherland, Toby Baker, Simon Shum, Olga C. Santos, Mercedes Rodrigo, Mutlu Cukurova, Ig Bittencourt, and Kenneth Koedinger. Ethics of AI in Education: Towards a Community-Wide Framework. *International Journal of Artificial Intelligence in Education*, 32, 04 2021. doi:10.1007/s40593-021-00239-1.

[9] Apple Inc. Siri, 2024. URL: https://www.apple.com/mx/siri/.

[10] Amazon. Alexa, 2024. URL: https://developer.amazon.com/en-US/alexa.

[11] Google. Google Assistant, 2024. URL: https://assistant.google.com/.

[12] Chinedu Wilfred Okonkwo and Abejide Ade-Ibijola. Chatbots applications in education: A systematic review. *Computers and Education: Artificial Intelligence*, 2:100033, 2021. doi:10.1016/j.caeai.2021.100033.

[13] UNESCO. Guidance for Generative AI in Education and Research. Technical report, Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura, París, 2023. ISBN: 978-92-3-100612-8. doi:10.54675/EWZM9535.

[14] CEPAL and UNESCO. La educación en tiempos de la pandemia de COVID-19. Technical report, Comisión Económica para América Latina y el Caribe y Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura, Santiago de Chile, 2020. URL: https://www.cepal.org/es/publicaciones/45904.

[15] UNESCO. Informe GEM 2023: Tecnología en la educación: ¿una herramienta en los términos de quién? Technical report, Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura, París, 2023. doi:10.54676/IDQE8212.

[16] John Sweller, Paul Ayres, and Slava Kalyuga. *Cognitive Load Theory*. Springer, 2011. ISBN: 978-1441981264. doi:10.1007/978-1-4419-8126-4.

[17] Asamblea General de la ONU. Transformar nuestro mundo: la Agenda 2030 para el Desarrollo Sostenible, 2015. URL: https://unctad.org/system/files/official-document/ares70d1_es.pdf.

[18] CEPAL. *Panorama Social de América Latina y el Caribe 2022: La transformación de la educación como base para el desarrollo sostenible*. Naciones Unidas, Santiago de Chile, 2022. ISBN: 9789211220957. URL: https://hdl.handle.net/11362/48518.

[19] UNICEF. Políticas de IA para la Infancia 2.0. Technical report, Fondo de las Naciones Unidas para la Infancia, 2021. URL: https://www.scribd.com/document/710672131/UNICEF-Global-Insight-policy-guidance-AI-children-2-0-2021-ES.

[20] Citlalli Selene Avalos Montiel. Modelo de navegación web usando voz, basado en contexto. Tesis de maestría, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2023. Departamento de Computación. URL: https://wwwcs.cs.cinvestav.mx/tesisgraduados/2023/resumenCitlalliAvalos.html.

[21] David Baidoo-Anu and Leticia Osei Ansah. Education in the Era of Generative Artificial Intelligence (AI): Understanding the Potential Benefits of ChatGPT in Promoting Teaching and Learning. *Journal of AI*, 7(1):52–62, 2023. doi:10.61969/jai.1337500.

[22] Eleni Adamopoulou and Lefteris Moussiades. Chatbots: History, Technology, and Applications. *Machine Learning with Applications*, 2:100006, 2020. doi:10.1016/j.mlwa.2020.100006.

[23] Michael J. Wooldridge. *An Introduction to Multiagent Systems*. Wiley, Chichester, 2nd edition, 2009. ISBN: 978-0-470-51946-2.

[24] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, and Tao Gui. The rise and potential of large language model based agents: a survey. *Science China Information Sciences*, 68:121101, 01 2025. doi:10.1007/s11432-024-4222-0.

[25] Alan Dix, Janet Finlay, Gregory Abowd, and Russell Beale. *Human-Computer Interaction*. Prentice Hall, 3rd edition, 2003. ISBN: 978-0130461094.

[26] Bayan Abu Shawar and Eric Atwell. Chatbots: Are They Really Useful? *Journal for Language Technology and Computational Linguistics*, 22(1):29–49, 2007. doi:10.21248/jlcl.22.2007.88.

[27] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. Stanford University, 2024. Draft disponible en línea de la 3ra edición. URL: https://web.stanford.edu/~jurafsky/slp3/.

[28] Sarah Olive, Kohei Uchimaru, Adele Lee, and Rosalind Fielding. *Shakespeare in East Asian Education*. Global Shakespeares Ser. Springer Nature, Cham, 2021. ISBN: 978-3-030-64795-7. doi:10.1007/978-3-030-64796-4.

[29] Rasa Technologies. Rasa Open Source Documentation, 2024. URL: https://rasa.com.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. URL: https://dl.acm.org/doi/10.5555/3295222.3295349?ref=mlq-ai.

[31] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, and Angela Fan. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024. doi:10.48550/arXiv.2407.21783.

[32] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, nov 2019. Association for Computational Linguistics. doi:10.18653/v1/D19-1410.

[33] ISO. Systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models. Standard, International Organization for Standardization, 2023. ISO/IEC 25010:2023. URL: https://www.iso.org/standard/78176.html.

[34] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch-Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. *Microservices: Yesterday, Today, and Tomorrow*, pages 195–216. 05 2017. ISBN: 978-3-319-67424-7. doi:10.1007/978-3-319-67425-4_12.

[35] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems*. CreateSpace Independent Publishing Platform, 3rd edition, 2017. ISBN: 978-1543057386.

[36] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2nd edition, 2021.

[37] Claus Pahl. Containerization and the PaaS Cloud. *IEEE Cloud Computing*, 2(3):24–31, 2015. doi:10.1109/MCC.2015.51.

[38] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. Technical Report Special Publication 800-145, National Institute of Standards and Technology (NIST), 2011. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[39] Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski. *Cloud Computing: Principles and Paradigms*. John Wiley & Sons, 2011. ISBN: 978-0-470-88799-8.

[40] Reza Shokri and Vitaly Shmatikov. Privacy-Preserving Deep Learning. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1310–1321, 2015. doi:10.1145/2810103.2813687.

[41] Ollama, Inc. Ollama Documentation and API Reference, 2024. URL: https://docs.ollama.com/.

[42] Docker Inc. Docker Documentation: Containerization and Docker Compose, 2024. URL: https://docs.docker.com/.

[43] CrewAI Inc. CrewAI Official Docs, 2024. URL: https://docs.crewai.com.

[44] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. AutoGen: Enabling Next-Generation LLM Applications with Reliable Multi-Agent Conversation. *arXiv preprint arXiv:2308.08155*, 2023. doi:10.48550/arXiv.2308.08155.

[45] LangChain Inc. LangChain Documentation: Agents and LangGraph, 2024. URL: https://docs.langchain.com/.

[46] Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative Agents for "Mind.[Ex]ploration of Large Language Model Society. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 51991–52008. Curran Associates, Inc., 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/a3621ee907def47c1b952ade25c67698-Paper-Conference.pdf.

[47] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating Multi-Agent Collaboration and Exploring Emergent Behaviors. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 20094–20136, 2024. URL: https://proceedings.iclr.cc/paper_files/paper/2024/file/578e65cdee35d00c708d4c64bce32971-Paper-Conference.pdf.

[48] Harrison Chase. LangChain: Building Applications with Large Language Models through Composability, 2022. URL: https://github.com/hwchase17/langchain.

[49] LangChain Inc. LangGraph: Stateful Multi-Agent Workflows with Language Models, 2024. URL: https://docs.langchain.com/oss/python/langgraph/.

[50] Fabio Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, 2007.

[51] Microsoft. Semantic Kernel: Integrate cutting-edge LLM technology quickly and easily into your apps, 2023. URL: https://github.com/microsoft/semantic-kernel.

[52] LlamaIndex. LlamaAgents: Async Service Oriented Architecture for Multi-Agent Systems, 2024. URL: https://github.com/run-llama/llama-agents.

[53] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, zili wang, Steven Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In B. Kim,

Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 23247–23275, 2024. URL: https://proceedings.iclr.cc/paper_files/paper/2024/file/6507b115562bb0a305f1958ccc87355a-Paper-Conference.pdf.

[54] LangChain Inc. State of AI Agents Report. Technical report, LangChain, 2025. URL: https://www.langchain.com/stateofaiagents.

[55] REACT: Synergizing Reasoning and Acting in Language Models. 2023. Publisher Copyright: © 2023 11th International Conference on Learning Representations (ICLR 2023). All rights reserved. Conference held from May 1–5, 2023.

[56] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL: https://openreview.net/forum?id=ehfRiF0R3a.

[57] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. URL: https://dl.acm.org/doi/abs/10.5555/3495724.3496517.

[58] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST'23, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3586183.3606763.

[59] MCP. Model Context Protocol (MCP): A Standard Interface for Tool and Data Access by AI Agents. Technical report, 2024. URL: https://modelcontextprotocol.io.

[60] Reid G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, 29(12):1104–1113, 1980. doi:10.1109/TC.1980.1675516.

[61] FIPA ACL Message Structure Specification. Technical report, Foundation for Intelligent Physical Agents, 2002. URL: https://www.yumpu.com/en/document/view/35150485/fipa-acl-message-structure-specification.

[62] Open Voice Interoperability Initiative. Open Floor Protocol (OFP) Specification v1.1. Technical report, Linux Foundation AI & Data, 2024. URL: https://openfloor.dev/.

[63] IEEE 2863-2026: Approved Draft Recommended Practice for Organizational Governance of Artificial Intelligence. Technical report, IEEE Standards Association, 2026. URL: https://standards.ieee.org/ieee/2863/10142/.

[64] Agent2Agent Working Group. Agent2Agent (A2A) Protocol Specification. Technical report, 2024. URL: https://a2a-protocol.org.

[65] Agent Communication Protocol Working Group. Agent communication protocol documentation, 2024. URL: https://agentcommunicationprotocol.dev/.

[66] Agent Network Protocol Working Group. Agent Network Protocol (ANP): Decentralized Networking for Cooperative AI Agents. Technical report, 2024. URL: https://agentnetworkprotocol.org.

[67] Google. Protocol Buffers: Language-neutral, platform-neutral extensible mechanism for serializing structured data, 2024. URL: https://protobuf.dev/overview/.

[68] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[69] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2004. ISBN: 978-0321200686.

[70] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014. URL: https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment.

[71] 1EdTech Consortium. Learning Tools Interoperability (LTI) Core Specification v1.3. Technical report, 1EdTech Consortium (formerly IMS Global), 2019. URL: https://www.imsglobal.org/spec/lti/v1p3/.

[72] Alex Büchner. *Moodle 4 Administration: An Administrator's Guide to Configuring, Securing, Customizing, and Extending Moodle*. Packt Publishing, Birmingham, UK, 4 edition, 2022. ISBN: 9781801816724. URL: https://www.packtpub.com/en-us/product/moodle-4-administration-9781801816724.

[73] Advanced Distributed Learning (ADL). Experience API (xAPI) Version 1.0.0. Technical report, Academic Advanced Distributed Learning Co-Lab, 2013. URL: https://github.com/adlnet/xAPI-Spec.

[74] OpenAI. Hello GPT-4o, 2024. URL: https://openai.com/index/hello-gpt-4o/.

[75] Gemini Team. Gemini 1.5: Unlocking Multimodal Understanding Across Millions of Tokens of Context. *arXiv preprint arXiv:2403.05530*, 2024. doi:10.48550/arXiv.2403.05530.

[76] Anthropic. The Claude 3.5 Sonnet Model, 2024. URL: https://www.anthropic.com/news/claude-3-5-sonnet.

[77] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc., 2022. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

[78] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: your language model is secretly a reward model. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS'23, Red Hook, NY, USA, 2023. Curran Associates Inc. URL: https://dl.acm.org/doi/abs/10.5555/3666122.3668460.

[79] An Yang, Jianbo Ba, Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Jianhui Dang, Xiaodong Deng, Yang Fan, and Wenbin Ge. Qwen2 Technical Report. *arXiv preprint arXiv:2407.10671*, 2024. doi:10.48550/arXiv.2407.10671.

[80] Georgi Gerganov and the llama.cpp contributors. GGUF: The Official Format for llama.cpp Models, 2024. URL: https://github.com/ggerganov/ggml/blob/master/docs/gguf.md.

[81] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Guangxuan Xiao, and Song Han. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. *GetMobile: Mobile Comp. and Comm.*, 28(4):12–17, January 2025. doi:10.1145/3714983.3714987.

[82] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. URL: https://dl.acm.org/doi/10.5555/3600270.3602468.

[83] Georgi Gerganov. llama.cpp: Port of facebook's LLaMA model in c/c++, 2023. URL: https://github.com/ggerganov/llama.cpp.

[84] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. SEI Series in Software Engineering. Addison-Wesley Professional, Boston, MA, 4th edition, 2021. ISBN: 978-0-13-688609-9.

[85] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, Boston, MA, 1994. ISBN: 978-0201633610.

[86] Scott Chacon and Ben Straub. *Pro Git*. Apress, Berkeley, CA, 2nd edition, 2014. ISBN: 978-1484200773. URL: https://git-scm.com/book/en/v2.

[87] Paul Leach, Michael Mealling, and Rich Salz. A Universally Unique IDentifier (UUID) URN Namespace. Technical Report RFC 4122, Internet Engineering Task Force (IETF), 2005. URL: https://datatracker.ietf.org/doc/html/rfc4122.

[88] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002. ISBN: 978-0321127426.

[89] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of JSON Schema. *Proceedings of the 25th International Conference on World Wide Web*, pages 263–273, 2016. doi:10.1145/2872427.2883029.

[90] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. In *Proceedings of the 30th Conference on Pattern Languages of Programs*, PLoP'23, USA, 2023. The Hillside Group. ISBN: 978-1-941652-19-0. URL: https://dl.acm.org/doi/10.5555/3721041.3721046.

[91] Edsger W Dijkstra. On the Role of Scientific Thought. In *Selected Writings on Computing: A Personal Perspective*, pages 60–66. Springer, 1982. doi:10.1007/978-1-4612-5695-3_11.

[92] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119, Red Hook, NY, USA, 2013. Curran Associates Inc. URL: https://dl.acm.org/doi/10.5555/2999792.2999959.

[93] Amit Singhal. Modern Information Retrieval: A Brief Overview. *IEEE Data Engineering Bulletin*, 24(4):35–43, 2001. URL: https://www.cs.columbia.edu/~gravano/Qual/Papers/singhal.pdf.

[94] Leslie Lamport. *LaTeX: A Document Preparation System: User's Guide and Reference Manual*. Addison-Wesley, Boston, 2nd edition, 1994. ISBN: 978-0201529838.

[95] Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall/Pearson Education, Upper Saddle River, NJ, 2003. ISBN: 978-0135974445.

[96] Chris Richardson. *Microservices Patterns: With Examples in Java*. Manning Publications, 2018. ISBN: 978-1617294549.

[97] Matthias Niedermaier, Florian Fischer, and Alexander von Kurnatowski. Observability in IT Infrastructures. *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pages 1–6, 2019. doi:10.1145/3339252.3340502.

[98] Qwen Team, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2.5 technical report, 2024. URL: https://arxiv.org/abs/2412.15115, arXiv:2412.15115.

[99] Mistral AI. Mistral nemo. https://mistral.ai/news/mistral-nemo/, 2024. Publicado el 18 de julio de 2024; en colaboración con NVIDIA.

[100] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size, 2024. URL: https://arxiv.org/abs/2408.00118, arXiv:2408.00118.

[101] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models, 2024. URL: https://arxiv.org/abs/2407.21783, arXiv:2407.21783.

[102] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008. ISBN: 978-0521865715. URL: https://nlp.stanford.edu/IR-book/, doi:10.1017/CBO9780511809071.

[103] Yujia Qin, Shengding Hu, Yankai Lin, Weize Miao, Ning Ding, Rui Geng, Haoze Fang, Yinpei Zheng, Zhiyuan Du, Baobao Yan, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023. URL: https://arxiv.org/abs/2307.16789, arXiv:2307.16789.

[104] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating Text Generation with BERT. In *International Conference on Learning Representations (ICLR)*, 2020. URL: https://arxiv.org/abs/1904.09675, arXiv:1904.09675.

[105] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large Language Model Connected with Massive APIs. *arXiv preprint arXiv:2305.15334*, 2023. URL: https://arxiv.org/abs/2305.15334, arXiv:2305.15334.

[106] Sandra G. Hart and Lowell E. Staveland. Development of NASA-TLX (task load index): Results of empirical and theoretical research. In *Human Mental Workload*, volume 52 of *Advances in Psychology*, pages 139–183. Elsevier, 1988. doi:10.1016/S0166-4115(08)62386-9.

[107] Sandra G. Hart. NASA-Task Load Index (NASA-TLX); 20 years later. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 50, pages 904–908. Sage Publications Sage CA: Los Angeles, CA, 2006. doi:10.1177/154193120605000909.

[108] Kawin Ethayarajh. How contextual are contextualized word representations? comparing the geometry of BERT, ELMo, and GPT-2 embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 55–65, Hong Kong, China, November 2019. Association for Computational Linguistics. URL: https://aclanthology.org/D19-1006, doi:10.18653/v1/D19-1006.

[109] Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945. doi:10.2307/3001968.

[110] W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, New York, 3rd edition, 1999. ISBN: 978-0471160687.

[111] Andy Field. *Discovering Statistics Using IBM SPSS Statistics*. Sage Publications, London, 4th edition, 2013. ISBN: 978-1446249185.