MANUAL TÉCNICO DEL SISTEMA PARA LA RESOLUCIÓN DE PROBLEMAS DE PROGRAMACIÓN NO LINEAL MEDIANTE INTEGRACIÓN DE MODELOS LLM Y MÓDULOS DE OPTIMIZACIÓN

Juan Carlos Barrera Guevara
Universidad de los Llanos
jc.bguevara@unillanos.edu.co

Diego Alejandro Machado Tovar
Universidad de los Llanos
damachado@unillanos.edu.co

Jesús Gregorio Delgado
Universidad de los Llanos
jg.delgado@unillanos.edu.co

ABSTRACT

El proyecto presenta una plataforma educativa para la resolución asistida de problemas de Programación No Lineal, implementada con Django y Channels. El flujo integra detección automática del método óptimo según la estructuración del problema, invocando módulos especializados —lagrangiano, condiciones KKT y programación cuadrática— con soporte para visualización progresiva vía WebSockets. Se enfatiza la modularidad del back-end, la trazabilidad de los cálculos y la generación pedagógica de respuestas, orientando el sistema tanto a la enseñanza como a la experimentación reproducible en entornos de optimización avanzada.

1 INTRODUCCIÓN

El sistema aborda la necesidad de convertir enunciados expresados en lenguaje natural en modelos matemáticos rigurosos susceptibles de resolución mediante técnicas de PNL. Se adopta un enfoque conversacional para reducir barreras de entrada, aprovechando un LLM externo que actúa como intérprete semántico. La motivación se fundamenta en la creciente complejidad de los problemas en investigación de operaciones y en la demanda de herramientas pedagógicas que combinen interacción natural con rigor numérico. El proyecto se alinea con la tendencia hacia sistemas híbridos que integran inteligencia artificial generativa y solvers especializados.

2 ALCANCE Y LIMITACIONES

Alcance: Interpretación automática de enunciados textuales; modelado en forma estándar (función objetivo, variables, restricciones); resolución mediante módulos internos (KKT, gradiente descendente, programación cuadrática, métodos adicionales documentados); interacción conversacional vía canal WebSocket; trazabilidad completa del pipeline.

Limitaciones: Dependencia de la API de Groq para la interpretación; cobertura de clases de problemas restringida a aquellos compatibles con los solvers implementados; sensibilidad a enunciados ambiguos; no se ejecutan métodos de optimización global; no se incorpora soporte nativo para problemas mixtos enteros; el desempeño depende de la conectividad y latencia del servicio externo.

3 ARQUITECTURA GENERAL DEL SISTEMA

La arquitectura se compone de un cliente conversacional (interfaz web con WebSockets), un backend Django con Channels, un módulo de integración Groq-Llama y componentes de optimización. El

flujo abarca captura del enunciado, envío al LLM, recepción y validación del modelo matemático, selección del solver, ejecución numérica y devolución del resultado.

Diagrama Textual de Arquitectura

- **Cliente Web Conversacional:** Aplicación desarrollada en React/HTML que emite mensajes y recibe actualizaciones en tiempo real mediante WebSocket.

- **Django Channels Layer:** El módulo `consumers_ai.py` gestiona la sesión del usuario, enruta mensajes hacia el backend síncrono o asíncrono y emite eventos de progreso.

- **Servicio de Interpretación:** El módulo `llm_client.py` compone prompts a partir de las instrucciones en archivos `.md` y del enunciado proporcionado. Envía la solicitud a la API de Groq (modelo Llama) y recibe la representación matemática generada.

- **Validador y Constructor de Modelo:** El módulo `model_builder.py` analiza la respuesta del modelo, verifica la coherencia de variables, parámetros, dominios y restricciones, y persiste el modelo formal estructurado.

- **Detector de Método:** El módulo `method_detector.py` clasifica el problema según su estructura (convexo, cuadrático, restricciones de igualdad o inequidad) y sugiere el solver apropiado.

- **Módulos de Optimización:** Módulos tales como `solver_kkt.py`, `solver_gradiente.py`, `solver_cuadratico.py`, `solver_lagrange.py`, entre otros, ejecutan los algoritmos numéricos necesarios para resolver el problema.

- **Gestor de Resultados:** Los módulos `views.py` y `consumers_ai.py` consolidan las soluciones finales, estadísticas y mensajes de carácter pedagógico, retornándolos al cliente conversacional.

- **Persistencia y Logging:** Uso de una base de datos relacional para almacenar historiales, registros de interpretación y auditorías de los cálculos realizados.

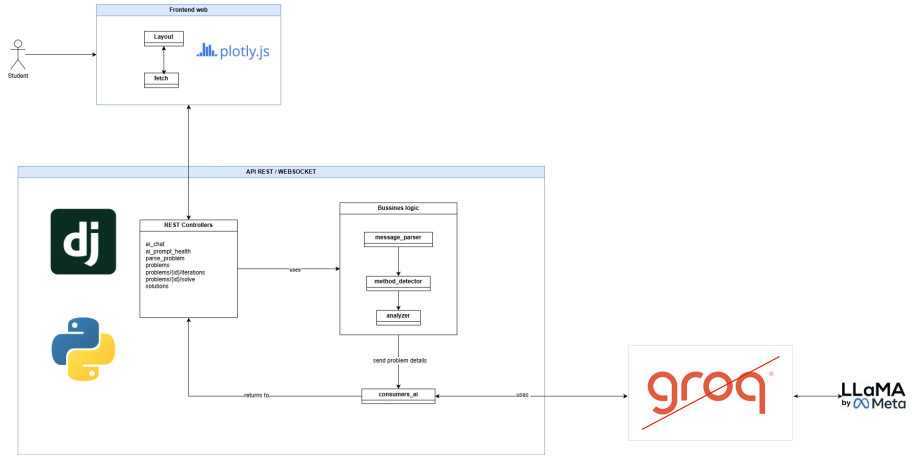

Figure 1: Arquitectura general del sistema

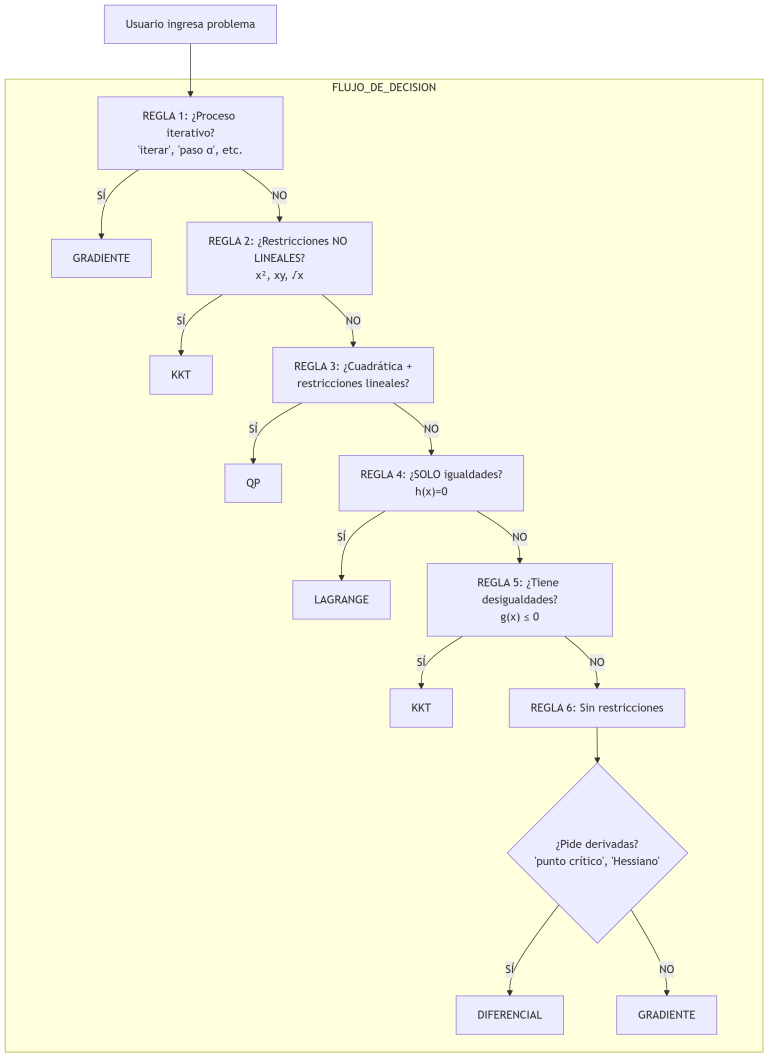

Figure 2: Diagrama de flujo para selección del método

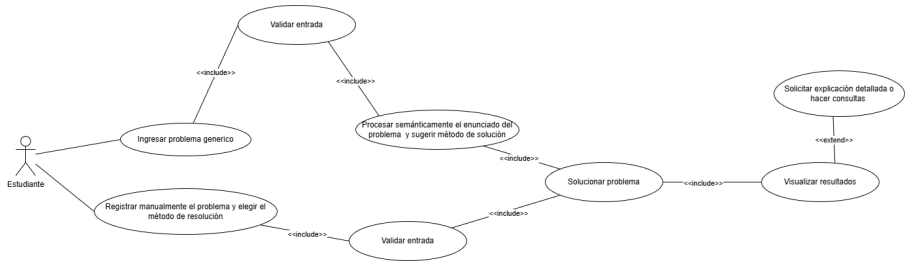

Figure 3: Diagrama de casos de uso

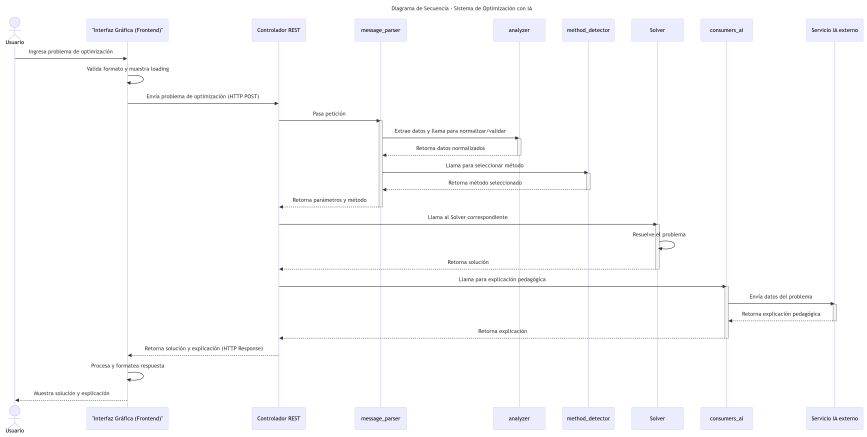

Figure 4: Diagrama de secuencia

4 Extracción del Modelo Matemático

El proceso inicia con la construcción de prompts controlados que combinan el enunciado del usuario con plantillas en Markdown que definen formato de salida, criterios de nomenclatura y ejemplos. El cliente Groq envía la petición al modelo Llama configurado con temperatura baja para priorizar precisión. La respuesta se recibe en JSON estructurado con secciones para variables, parámetros, función objetivo y restricciones. El módulo de validación realiza:

- Consistencia de índices, dominios y unidades.
- Verificación de que toda variable presente en las restricciones aparezca también en la función objetivo, y viceversa.
- Normalización de la notación matemática, incluyendo la conversión a sintaxis simbólica estándar cuando corresponde.
- Introducción de confirmaciones al usuario en los casos donde se detecten ambigüedades en el enunciado o en la formulación propuesta.
- Una vez aprobada la representación, esta se traduce a objetos internos del sistema (por ejemplo, `OptimizationProblem`), los cuales quedan listos para ser procesados por los solvers.

5 Especificación Interna del Solver

- **Condiciones KKT (`solver_kkt.py`):** Asume diferenciabilidad y convexidad local. Calcula gradientes y matrices Hessianas, construye el sistema aumentado y lo resuelve mediante descomposición LU. Verifica condiciones de regularidad (LICQ) y factibilidad primal-dual.
- **Gradiente Descendente (`solver_gradiente.py`):** Orientado a problemas sin restricciones o con penalización. Utiliza pasos adaptativos mediante búsqueda lineal de Armijo, detiene el proceso según la norma del gradiente y controla oscilaciones mediante amortiguamiento.
- **Programación Cuadrática (`solver_cuadratico.py`):** Diseñado para funciones objetivo cuadráticas con restricciones lineales. Realiza reducción a forma estándar y resuelve empleando un algoritmo de punto interior simplificado o un método activo, dependiendo de la dimensión del problema.
- **Lagrangiano (`solver_lagrange.py`):** Adecuado para problemas con restricciones de igualdad. Construye los multiplicadores de Lagrange y resuelve el sistema resultante.

- Cada módulo documenta los supuestos, el dominio de aplicabilidad y las condiciones de convergencia. Asimismo, los algoritmos registran iteraciones, tolerancias y fallas en las bitácoras del sistema.

6 Trazabilidad del Desarrollo

- **Decisiones Arquitectónicas:** Se adoptó Django junto con Channels para combinar la robustez del modelo HTTP con capacidades de comunicación en tiempo real.
- **Selección del LLM:** Se seleccionó la API de Groq con el modelo Llama debido a su baja latencia y a su compatibilidad con prompts estructurados.
- **Diseño de Módulos de Optimización:** La modularidad favorece la ejecución de pruebas aisladas y la extensibilidad del sistema. Se establecieron convenciones internas de interfaz, como `solve(problem, options)`, para uniformar la interacción entre módulos.
- **Instrucciones `.md`:** Los documentos base definen el formato y las políticas de interpretación, lo que facilita el control de las salidas generadas y los procesos de auditoría.
- **Pipeline:** Se implementó una cadena determinista desde la adquisición del texto hasta la ejecución del solver, incorporando etapas de validación y registro de eventos.
- **Criterios de Evaluación:** Se consideran la precisión del modelo interpretado, el tiempo total de respuesta, la convergencia del solver y la alineación con soluciones de referencia.

7 API Interna y Endpoints Relevantes

- **POST /api/problems/parse/:** Recibe enunciados en lenguaje natural, invoca el LLM y devuelve el modelo preliminar generado.
- **WebSocket /ws/solver/:** Canal principal para el intercambio conversacional. Permite el envío de pasos intermedios, retroalimentación del proceso y resultados finales.
- **GET /api/methods/:** Retorna la lista de métodos de resolución disponibles junto con sus metadatos y condiciones de aplicabilidad.
- **POST /api/solve/:** Acepta un modelo formal en formato JSON y los parámetros de ejecución. Activa el solver seleccionado y gestiona la ejecución del algoritmo.

8 Flujo de Ejecución Detallado

- El usuario introduce el enunciado del problema directamente en el chat del sistema.
- El consumidor WebSocket transmite el texto al backend para su procesamiento inicial.
- El backend construye un prompt utilizando las instrucciones contenidas en los archivos `.md` y envía la solicitud de interpretación al modelo Llama a través de la API de Groq.
- La respuesta estructurada es validada y normalizada; si se detectan ambigüedades o inconsistencias, se generan solicitudes de aclaración al usuario.
- El detector de método analiza el perfil del problema y sugiere el solver adecuado. El usuario puede confirmar la recomendación o seleccionar un método alternativo.
- El solver seleccionado ejecuta las iteraciones correspondientes, registra métricas relevantes y produce la solución numérica del modelo.
- El backend formatea los resultados, incorpora análisis de convergencia y añade mensajes de carácter pedagógico sobre el proceso.
- El cliente recibe actualizaciones en tiempo real y finalmente accede al resultado completo, incluyendo la opción de descargar un reporte.

9 Consideraciones de Rendimiento y Estabilidad

- Cacheo de prompts e interpretaciones parciales para evitar reenvíos redundantes y reducir la latencia general del sistema.

- Control de tiempos de espera mediante políticas de reintentos exponenciales dirigidas a la API de Groq, mitigando fallos transitorios de red.
- Validación numérica continua a través de pruebas de factibilidad y evaluaciones de la función objetivo, garantizando coherencia en la representación matemática.
- Manejo robusto de errores, capturando excepciones de red, inconsistencias en el modelo interpretado y posibles divergencias del solver. Se generan mensajes claros y accionables para el usuario.
- Monitoreo de recursos del servidor y utilización de colas de procesamiento para balancear sesiones concurrentes y preservar la estabilidad.

10 Pruebas Realizadas

- **Casos Sintéticos:** Se emplearon problemas convexos y no convexos diseñados para validar la precisión de las interpretaciones y detectar fallas sistemáticas.
- **Comparación con Soluciones de Referencia:** Se utilizaron benchmarks provenientes de libros de texto y se resolvieron mediante herramientas externas como AMPL y MATLAB, con el fin de verificar exactitud numérica y consistencia estructural.
- **Pruebas de Estrés:** Se ejecutaron cargas concurrentes con múltiples sesiones activas para medir latencia, uso de memoria y comportamiento del sistema bajo demanda elevada.
- **Pruebas de Regresión:** Se desarrolló una suite automatizada que garantiza estabilidad frente a cambios en los documentos de instrucciones `.md` o en los módulos de solver.

Los resultados obtenidos muestran una consistencia superior al 90 por ciento en interpretaciones estructurales y una convergencia dentro de tolerancias predefinidas para los problemas compatibles.

11 Manual de Instalación y Despliegue

Prerequisitos

Python 3.11 o superior, Node.js para el frontend (opcional) y PostgreSQL o SQLite como motor de base de datos.

Entorno Virtual

```
python -m venv venv
venv\Scripts\activate
```

Dependencias

Instalar todas las dependencias del proyecto:

```
pip install -r requirements.txt
```

Variables de Entorno

Configurar las siguientes variables según el entorno de despliegue:

- `GROQ_API_KEY`
- Parámetros de la base de datos (host, usuario, contraseña, puerto)
- Ajustes de Channels y backend de WebSockets (Redis si corresponde)

MIGRACIONES

Aplicar las migraciones iniciales:

```
python manage.py migrate
```

COLECCIÓN DE ARCHIVOS .MD

Verificar que los documentos de instrucciones se encuentren en:

```
opti_app/prompts/
```

EJECUCIÓN EN DESARROLLO

Iniciar el servidor ASGI en entorno local:

```
python -m daphne opti_learn.asgi:application --bind 127.0.0.1 --port 8000
```

DESPLIEGUE EN PRODUCCIÓN

Utilizar un servidor ASGI como Daphne o Uvicorn detrás de Nginx para manejo de tráfico HTTP y WebSocket. Configurar supervisión de procesos mediante `systemd` u otro gestor y habilitar certificados TLS según el entorno.

12 GUÍA DE MANTENIMIENTO Y EXTENSIBILIDAD

- **Agregar Nuevos Métodos:** Implementar el módulo correspondiente en `core/solvers/`, registrarlo en `method_detector.py`, documentarlo en el catálogo de métodos y añadir los casos de prueba pertinentes.
- **Actualizar Instrucciones del LLM:** Editar los archivos `.md`, versionar los cambios realizados y ejecutar pruebas de regresión para garantizar estabilidad.
- **Mejorar la Interpretación Semántica:** Ajustar los prompts, incorporar ejemplos de referencia e incluir filtros de postprocesamiento para refinar la representación matemática producida.
- **Ampliar el Backend:** Seguir los patrones establecidos en las vistas y consumidores existentes, mantener los contratos JSON y actualizar la documentación respectiva en Swagger u OpenAPI.
- **Observabilidad:** Revisar periódicamente los logs estructurados, las métricas expuestas mediante Prometheus y las alertas asociadas a disponibilidad y rendimiento.

13 CONCLUSIONES TÉCNICAS

El sistema presenta una arquitectura híbrida que separa la interpretación semántica del enunciado y la resolución numérica del modelo, lo que permite precisión en la formulación y flexibilidad en la optimización. La integración con Groq-Llama agiliza la conversión de lenguaje natural a estructuras matemáticas formales, mientras que los solvers internos ofrecen control total sobre los algoritmos, sus supuestos y su trazabilidad. Se proyectan futuras iteraciones orientadas a ampliar la cobertura de métodos de optimización, mejorar el manejo de ambigüedades y optimizar el rendimiento en condiciones de alta demanda. El proyecto demuestra la viabilidad de integrar LLMs con motores especializados para aplicaciones avanzadas dentro del campo de la investigación de operaciones.