Maestría en Inteligencia Artificial Aplicada

Proyecto Integrador

Avance 5: Modelo Final

Profesores:
Dra. Grettel Barceló Alonso
Dr. Luis Eduardo Falcón
Dr. Guillermo Mota Medina

Equipo 59

A01795457	Renzo Antonio Zagni Vestrini
A01362405	Roger Alexei Urrutia Parke
A01795501	Héctor Raúl Solorio Meneses

Contenido

Introduccion

El presente documento constituye el Avance 5 del proyecto de investigación centrado en el desarrollo de un sistema *Agentic AI* capaz de generar aplicaciones completas de base de datos a partir de descripciones en lenguaje natural.

En esta etapa se introdujo y consolidó la arquitectura del orquestador, diseñada para coordinar el flujo de comunicación entre el usuario, los modelos de lenguaje y los módulos de validación, garantizando la trazabilidad y el control en todo el proceso generativo. Esta arquitectura permite transformar las descripciones textuales en artefactos ejecutables —base de datos, backend y frontend— a través de una secuencia estructurada de generación, verificación y ensamblaje.

El Avance 5 se enfocó en aplicar la técnica de entonación fina de prompts como estrategia para abordar tres desafíos fundamentales:

1. Reducir la alucinación semántica y estructural de los modelos LLM.

2. Asegurar la coherencia interna entre los artefactos generados (base de datos, API y frontend).

3. Lograr consistencia de resultados entre distintos modelos y proveedores, de modo que la aplicación producida sea funcionalmente equivalente, sin depender del modelo específico utilizado.

Este último objetivo representa un avance significativo: al uniformar la estructura y semántica de los *prompts*, se logró que modelos de distinta arquitectura: GPT-5, GPT-5-mini, DeepSeek-Code y Claude-Opus generaran artefactos casi idénticos, demostrando que la entonación del lenguaje de instrucción puede desacoplar el sistema de un modelo en particular, preservando la independencia tecnológica del orquestador.

Entonación Fina de los Prompts

Los modelos de lenguaje de gran tamaño (LLM) son sistemas generativos que operan sobre probabilidades lingüísticas. Su flexibilidad conlleva un riesgo inherente: la variabilidad semántica y la alucinación estructural, es decir, la producción de información no solicitada o la inconsistencia entre componentes generados.

La entonación fina de prompts surge como una técnica no paramétrica de control lingüístico que busca reducir esa variabilidad. A través del diseño de las instrucciones, definiendo roles, formatos y restricciones; es posible inducir comportamientos más estables y replicables.

En el contexto de este proyecto, la entonación fina fue aplicada con un objetivo adicional: normalizar la salida entre modelos distintos, logrando así uniformidad funcional y semántica. Esta aproximación se basa en el principio de que, cuando el lenguaje de instrucción es suficientemente prescriptivo, los modelos tienden a converger hacia estructuras de salida idénticas, independientemente de sus diferencias internas.

Metodología

1. Arquitectura del orquestador

El orquestador, es el core del sistema y es el responsable de la gestion del proceso de generación mediante tres etapas principales:

1. Análisis del requerimiento, donde se interpreta el texto del usuario y se genera un esquema inicial de datos.

2. Refinamiento del esquema, que ajusta la estructura según la retroalimentación proporcionada.

3. Generación integral de artefactos, encargada de producir código ejecutable para backend, frontend y base de datos.

El orquestador mantiene control de estado, validación automática y registro persistente de cada generación, asegurando reproducibilidad y trazabilidad del proceso.

2. Entonación Fina de los Prompts

Los *prompts* empleados en el sistema fueron diseñados y entonados bajo tres principios:

- Estructura controlada, donde se especifica el formato de salida (JSON, Python, HTML).
- Lenguaje prescriptivo, que define librerías, convenciones y tipos de datos.
- Salida determinista, que prohíbe comentarios o explicaciones en la respuesta.

A continuación se presentan muestras parciales y representativas de los *prompts* utilizados en el sistema, seleccionadas para ilustrar la estructura y estilo de entonación empleados.

Prompt de análisis de requerimiento — `analysis_prompt`

Este *prompt* solicita al modelo analizar la descripción textual de la aplicación y producir un esquema estructurado en formato JSON con la entidad principal y sus campos.

Ejemplo:

> *"Analyze this application description and identify the main entity, its fields, and data types.*
> *Return a JSON object with:*
> *– entityName: string (PascalCase)*
> *– fields: array of {name, label, type, required}.*
> *Return ONLY valid JSON, no markdown or explanations."*

Objetivo: guiar al modelo hacia una respuesta estructurada y libre de texto libre.
Resultado esperado: un JSON válido, limpio y ejecutable.
Control de alucinación: uso de validación por expresiones regulares y reintentos automáticos de parseo.

Prompt de refinamiento — `refine_prompt`

Se aplica cuando el usuario desea modificar el esquema previamente generado. Este *prompt* instruye al modelo a conservar la estructura general y aplicar solo los cambios solicitados.

Ejemplo:

> *"Modify this database schema based on user feedback.*
> *Maintain the same structure and field consistency.*
> *Return ONLY valid JSON with the same format."*

Objetivo: mantener coherencia entre versiones del esquema.
Resultado esperado: estructura JSON corregida sin pérdida de integridad.

Prompt de generación de artefactos — `schema_prompt`, `api_prompt`, `frontend_prompt`, `template_prompt`

En esta fase, el sistema genera el código completo del proyecto, compuesto por el modelo de base de datos, el backend, la interfaz HTML y los archivos auxiliares.

Las siguientes son muestras de los *prompts* utilizados, la versión completa y actualizada de todos ellos se encuentra disponible en el repositorio del proyecto en GitHub

`schema_prompt`

> *"Write valid Python 3.11 SQLAlchemy 2.0 code for this entity.*
> *Use DeclarativeBase, mapped_column, and proper type mapping (string→String, number→Integer, etc.).*
> *Return ONLY the Python code, no markdown."*

`api_prompt`

> *"Generate a FastAPI backend implementing CRUD routes for the given entity.*
> *Include database session handling, imports, and error management.*
> *Ensure all endpoints are functional and coherent with the schema."*

`frontend_prompt`

> *"Create a FastAPI + Jinja2 frontend with HTML templates to display and manipulate records.*
> *Use semantic HTML5, responsive design, and clear labels for each field."*

`template_prompt`

> *"Generate the HTML template for CRUD management.*
> *Include form inputs for each field, validation attributes, and a responsive layout."*

Estos *prompts* fueron ajustados iterativamente hasta obtener salidas equivalentes entre modelos, reduciendo casi por completo la divergencia estructural o semántica.

3. Escenarios de Prueba

Se realizaron diez pruebas de generación de aplicaciones CRUD a partir de descripciones textuales equivalentes, procesadas por los cuatro modelos evaluados: GPT-5, GPT-5-mini, DeepSeek-Code y Claude-Opus.

Los artefactos generados fueron:

- Base de datos: esquema SQLAlchemy 2.0.
- Backend: servicios FastAPI con rutas CRUD.
- Frontend: interfaz HTML y lógica Jinja2.

Las observaciones incluyeron validez del código, coherencia entre módulos, tiempo de ejecución y similitud entre resultados intermodelos.

Resultados

Los resultados obtenidos en esta fase reflejan el impacto directo de la entonación fina de prompts sobre la estabilidad y la coherencia de los artefactos generados por los diferentes modelos de lenguaje.

La evaluación se realizó de manera comparativa, utilizando un conjunto controlado de escenarios idénticos para los cuatro modelos seleccionados. En cada caso se midió la validez funcional del código, la necesidad de intervención humana, el tiempo de generación y la similitud estructural entre los artefactos producidos.

Los hallazgos permiten observar una notable reducción de las alucinaciones semánticas y una convergencia casi total en la estructura de los resultados, confirmando la efectividad del enfoque propuesto para lograr consistencia intermodelo y reproducibilidad del proceso generativo.

Modelo	Ejecución sin errores	Intervención manual	Tiempo promedio	Observaciones
GPT-5	10/10	None	7 s	Generación estable y coherente.
GPT-5-mini	9/10	Leve	4 s	Salida simplificada, misma estructura.
DeepSeek-Code	10/10	None	6 s	Precisión técnica notable.
Claude-Opus	10/10	None	8 s	Excelente claridad y formato.

Tabla 1: Evaluación Funcional

Modelo	Estructura de archivos	Campos y tipos de datos	Estilo de código	Diferencias observadas	Variación
GPT-5	Idéntica	100%	Coincidente	Nulas	0%
GPT-5-mini	Idéntica	100%	Coincidente	Mínimas	0%
DeepSeek-Code	Idéntica	98%	Coincidente	Nulas	0.5%
Claude-Opus	Idéntica	100%	Coincidente	Nulas	0.2%

Tabla 2: Homogeneidad entre modelos

El resultado más destacable fue la uniformidad funcional observada entre los cuatro modelos evaluados. Todos generaron artefactos casi idénticos en estructura, comportamiento y formato, validando la hipótesis de que una entonación fina y cuidadosamente diseñada puede neutralizar la variabilidad intermodelo y conducir a sistemas equivalentes y reproducibles. Esta convergencia evidencia que el control lingüístico a nivel de *prompt* es un mecanismo efectivo para estabilizar la salida de modelos de distinta arquitectura, sin necesidad de ajustes paramétricos ni dependencias específicas de proveedor.

En conjunto, los resultados confirman que la estrategia aplicada que redujo las alucinaciones semánticas y estructurales y permitió alcanzar una consistencia transversal en la generación de código *full-stack*.

Selección del Modelo Final

La selección del modelo final se apoya en los resultados obtenidos en los Avances 4 y 5, los cuales, en conjunto, permiten establecer una evaluación integral del desempeño de los modelos de lenguaje utilizados en el sistema *Agentic AI Generator*.

El objetivo de esta etapa fue determinar cuál de los modelos evaluados ofrecía el mejor equilibrio entre calidad de generación, eficiencia operativa y consistencia estructural, tanto en condiciones de prompting estándar (Avance 4) como bajo entonación fina (Avance 5).

Evaluación Avance 4

Durante el Avance 4, se evaluaron siete modelos (GPT-5, GPT-5-mini, Claude-Opus, Claude-Sonnet, DeepSeek-Coder, GPT-4o-mini y GPT-4-Turbo) considerando métricas cuantitativas —tokens utilizados, costo promedio, tiempo de ejecución, líneas de código y desviación estándar de *runtime*— y cualitativas —calidad del código, estabilidad y relación costo/calidad.

Modelo	Tokens	Costo	Runtime (sec)	Lineas de Código	σ (Runtime)
GPT-5	31,823	$0.11504	475	1,104	139.99
GPT-5-mini	21,502	$0.00694	258.31	916	60.19
Claude-opus-4-1	15,872	$0.00612	200.65	875	31.25
Claude-sonnet-4-5	12,649	$0.00212	100.22	875	7.39
DeepSeek-coder	5,533	$0.00245	171.90	430	60.58
GPT-4o-mini	3,931	$0.00061	41.76	410	14.47
GPT-4-turbo	2,718	$0.00465	53.54	334	32.16

Tabla 3: Resultados promedio de evaluación de modelos LLM

En términos cualitativos, el modelo GPT-5 produjo las arquitecturas más completas y coherentes, mientras que GPT-5-mini logró resultados muy similares con una fracción del costo y menor tiempo de procesamiento.

Su código fue limpio, estructurado y sintácticamente correcto, mostrando una excelente correspondencia entre entidades, relaciones y componentes de la interfaz.

Modelo	Costo Promedio	Tiempo Promedio	Calidad de Código	Relación Costo/Calidad	Ideal para
GPT-5	Alto (~$0.038 USD)	Medio-alto	⭐⭐⭐⭐⭐	Alta	Aplicaciones empresariales complejas
GPT-5-mini	Bajo (~$0.016 USD)	Medio-bajo	⭐⭐⭐⭐	**Excelente**	Desarrollo iterativo y prototipado
Claude-Opus	Alto (~$0.047 USD)	Alto	⭐⭐⭐⭐⭐	Media	Casos con alta semántica y trazabilidad
Claude-Sonnet	Medio (~$0.037 USD)	Medio	⭐⭐⭐⭐	Buena	Modelos bien estructurados
GPT-4o-mini	Muy bajo (~$0.015 USD)	Bajo	⭐⭐⭐	Buena	Pruebas o generación rápida
DeepSeek-Coder	Muy bajo (~$0.012 USD)	Bajo	⭐⭐	Regular	Casos simples, bajo presupuesto

Tabla 4: Evaluación de los modelos

A partir de estos resultados, GPT-5-mini fue seleccionado preliminarmente como el modelo más eficiente y equilibrado, debido a su excelente relación costo-calidad, velocidad de inferencia y capacidad para mantener coherencia estructural en los artefactos generados.

Evaluación Avance 5

El Avance 5 tuvo como propósito confirmar esa elección mediante la aplicación de entonación fina de prompts, diseñada para controlar la alucinación y lograr consistencia entre modelos de distintos proveedores.

Se evaluaron cuatro modelos: GPT-5, GPT-5-mini, DeepSeek-Code y Claude-Opus, en diez escenarios de generación de aplicaciones CRUD idénticos.

Los resultados confirmaron una uniformidad funcional casi total entre los cuatro modelos evaluados.

La entonación fina permitió que todos los modelos generaran artefactos equivalentes en estructura, comportamiento y formato, reduciendo significativamente las alucinaciones semánticas.

En este contexto, GPT-5-mini se mantuvo como el modelo más eficiente, combinando velocidad, bajo costo y consistencia intermodelo.

Selección Final

Integrando los hallazgos de ambas fases, se concluye que GPT-5-mini es el modelo óptimo para el sistema *Agentic AI Generator*. El Avance 4 demostró su superioridad en términos de eficiencia costo-rendimiento y calidad estructural, mientras que el Avance 5 validó su capacidad para mantener coherencia y reproducibilidad aun bajo condiciones de prompting controlado.

GPT-5-mini ofrece:

- Estructura y semántica equivalentes a GPT-5 con menor costo (~80 % menos).
- Generación más rápida (4 s promedio).
- Baja variabilidad entre ejecuciones.
- Coherencia completa entre frontend, backend y base de datos.
- Estabilidad intermodelo validada por homogeneidad estructural.

En consecuencia, se ratifica la elección de GPT-5-mini como modelo final, consolidando una arquitectura agnóstica al proveedor, eficiente y lingüísticamente controlable, capaz de generar artefactos full-stack consistentes y reproducibles bajo cualquier entorno LLM.

Conclusión

El Avance 5 confirmó que la entonación fina de los prompts es una técnica efectiva para reducir la alucinación de los modelos de lenguaje y asegurar la generación coherente de los artefactos del sistema.

Al aplicar un diseño más estructurado en las instrucciones, los cuatro modelos evaluados: GPT-5, GPT-5-mini, DeepSeek-Code y Claude-Opus, generaron resultados prácticamente idénticos, demostrando que es posible mantener consistencia funcional sin depender del modelo o del proveedor.

La arquitectura del orquestador permitió controlar el proceso de generación, verificar los resultados y garantizar la coherencia entre la base de datos, el backend y el frontend. Con ello se consolidó un flujo estable, reproducible y trazable que reduce la intervención manual y mejora la calidad general del producto generado.

Finalmente, la selección de GPT-5-mini como modelo principal se mantiene justificada. Este modelo logró el mejor equilibrio entre costo, velocidad y calidad del código, ofreciendo resultados equivalentes a los de modelos más grandes con un uso menor de recursos. Su desempeño confirma que la entonación fina permite alcanzar una salida estable y coherente, independiente del modelo utilizado.

En conjunto, este avance demuestra que el control y la entonación de los prompts puede sustituir ajustes técnicos complejos, y que un esquema de prompting bien diseñado es suficiente para generar aplicaciones funcionales y consistentes de manera automática.