GPTur

Claudia Hernández Pérez, Joel Aparicio Tamayo, and Kendry Javier Del Pino Barbosa

Facultad de Matemática y Computación, La Habana, Cuba

Resumen. GPTur es un sistema diseñado con una arquitectura de Generación Aumentada por Recuperación (RAG) con agentes especializados para cada funcionalidad y una base de datos vectorial para la recuperación de la información. Posee, además, varios agentes encargados de tópicos específicos como gastronomía, vida nocturna, alojamiento y lugares históricos, que comparten datos en una pizarra central. Estos agentes están dirigidos por un guía y un planificador. El primero se encarga de responder consultas mientras el segundo utiliza algoritmos para planear itinerarios ajustados a presupuestos. Este agente resuelve un problema de satisfacción de restricciones utilizando la técnica de precocido simulado. También, el programa preprocesa la consulta para obtener ciertas entidades, palabras claves y sentimientos que puedan influir en los resultados esperados. Todo eso, permite conformar una respuesta acertada, acorde a las necesidades del usuario, validadas experimentalmente.

Palabras clave: RAG, agente, problema de satisfacción de restricciones, precocido simulado

1 Introducción

La industria turística enfrenta desafíos únicos en contextos con limitaciones en la infraestructura digital y actualización constante de servicios. Cuba, como destino turístico emblemático, requiere soluciones innovadoras que superen estas barreras mediante inteligencia artificial. *GPTur*, un sistema de agentes inteligentes que funciona como guía turístico virtual, está diseñado específicamente para ser una alternativa viable en ese escenario.

La aplicación consiste en un chatbot capaz de responder consultas sobre destinos turísticos cubanos y sus peculiaridades, así como de generar itinerarios para *tours* ajustándose a un presupuesto. El sistema está programado en *Python*, aprovechando la comodidad del lenguaje y la diversidad de bibliotecas útiles que ofrece. La interfaz está diseñada con *streamlit* (framework de *Python* para *frontend*), pues ofrece un estilo sencillo, moderno y fácil de programar para entornos web. El código fuente de la aplicación se puede encontrar en la plataforma GitHub: https://github.com/ClaudiaHdezPerez/GPTur.

2 Detalles de Implementación

En esta sección, se abordarán los distintos temas asociados a la implementación de la solución propuesta, que incluyen la arquitecura RAG y multiagente, funcionamiento del *crawler* automatizado y recopilación del corpus, base de datos vactorial y representación semántica del conocimiento mediante *embeddings*, procesamiento de lenguaje natural, algoritmos basados en metaheurísticas, entre otros.

2.1 Arquitectura RAG (*Generación Aumentada por Recuperación*)

Un sistema de Generación Aumentada por Recuperación combina generación de texto con recuperación de información para mejorar la calidad, precisión y actualidad de las respuestas obtenidas de los modelos de lenguaje. Está compuesto por una entrada, un recuperador, un generador y una salida. La implementación propuesta es bastante sencilla, ajustada a la definición:

1. Entrada: constituida por la consulta del usuario en lenguaje natural.
2. Recuperador: agente que recibe la consulta y le pide a la base de datos vectorial que recupere la información relacionada.
3. Generador: agente que utiliza la consulta y la información recuperada para decidir si la respuesta debe ser generada por el guía o el planificador. Para tomar esa decisión le pide a un modelo de lenguaje que detecte la intención del usuario.
4. Salida: respuesta generada por el agente inteligente encargado de elaborarla.

2.2 Arquitectura multiagente

Un sistema multiagente está constituido por un conjunto de agentes, cada uno con determinadas capacidades y recursos para la solución de problemas inherentemente distribuidos. Los agentes interactúan entre ellos, determinan y coordinan tareas a realizar según sus capacidades y recursos.
GPTur cuenta con dos agentes fundamentales: el guía y el planificador. Existen, además, otros especializados en distintos tópicos: gastronomía, vida nocturna, alojamiento y lugares históricos. Todos ellos

2.3 Crawler Automatizado en Dos Fases. Recopilación del *Corpus*

Se conoce como *crawler* a un software que realiza un proceso automatizado de navegar por la Web de manera sistemática para indexar y recopilar información de diferentes sitios web. El sistema implementa un *crawler* en dos fases:

- *Fase inicial:* Recopila, almacena y organiza datos en un repositorio vectorial, garantizando una gestión eficiente de los vectores de información.
- *Fase dinámica:* Utiliza automáticamente información externa cuando el sistema detecte respuestas incompletas o desactualizadas.

Fase Inicial: En esta fase el *crawler* se configura con los parámetros necesarios, como las URLs objetivo (disponibles en *src /data /sources.json*), criterios de búsqueda y profundidad de rastreo. Utiliza técnicas de *scraping* (extracción de la web) para navegar por sitios web turísticos, identificando y extrayendo información relevante como descripciones de lugares, eventos, alojamientos y actividades. Los datos extraídos son normalizados y almacenados en un formato *.json*, facilitando su posterior análisis y procesamiento.

Fase Dinámica: ...

Recopilación y Construcción del Corpus: Este proceso consta de cuatro tareas principales:

- *Selección de Fuentes:* Se identificaron y seleccionaron fuentes web relevantes para el turismo, como portales oficiales, blogs y sitios de reseñas.
- *Rastreo y Descarga:* El *crawler* recorre las páginas seleccionadas, descargando el contenido textual y metadatos asociados.
- *Filtrado y Normalización:* Se aplican filtros para eliminar información irrelevante o duplicada, y se normalizan los datos para mantener la coherencia en el corpus.
- *Almacenamiento:* El corpus final se almacena en formato *.json* (disponible en *src /data /processed /normalized_data.json*).

Este proceso automatizado permite mantener un corpus representativo del dominio turístico, esencial para el tema que aborda el sistema.

2.4 Base de Datos Vectorial y Modelo de Representación del Conocimiento

El sistema utiliza *ChromaDB* como base de datos vectorial. *ChromaDB* es una solución especializada para el almacenamiento y consulta de vectores de alta dimensión, optimizada para tareas de inteligencia artificial y procesamiento de lenguaje natural.

- *Integración con el Sistema:* La integración se realiza a través del módulo *src /vector_db /chroma_storage.py*, que proporciona funciones para insertar, actualizar y consultar vectores en la base de datos.
- *Almacenamiento Eficiente: ChromaDB* permite almacenar grandes volúmenes de *embeddings* generados a partir del corpus, manteniendo tiempos de respuesta bajos incluso en consultas complejas.
- *Búsqueda por Similitud:* Utilizando algoritmos de búsqueda aproximada de vecinos más cercanos (ANN), *ChromaDB* facilita la recuperación de fragmentos de información relevantes a partir de consultas vectorizadas, mejorando la precisión y relevancia de las respuestas del sistema.
- *Escalabilidad y Actualización: ChromaDB* soporta la actualización dinámica de la base de datos, permitiendo añadir o eliminar vectores conforme el corpus evoluciona, sin afectar el rendimiento general.

– *Persistencia y Seguridad:* Los datos almacenados en *ChromaDB* pueden persistirse en disco, asegurando la integridad y disponibilidad de la información incluso ante reinicios del sistema.

El uso de *ChromaDB* proporciona una infraestructura robusta y escalable para la gestión del conocimiento en forma vectorial, facilitando la integración de capacidades avanzadas de recuperación semántica y razonamiento en el sistema.

Modelo de Representación del Conocimiento: El sistema emplea un modelo de representación del conocimiento basado en *embeddings* semánticos, lo que permite capturar relaciones y similitudes entre conceptos turísticos. Las principales características son:

– *Representación Distribuida:* Cada entidad o fragmento de información se representa como un vector en un espacio de alta dimensión, facilitando la comparación y agrupación de conceptos similares.
– *Recuperación Semántica:* Las consultas del usuario se transforman en vectores y se comparan con los almacenados en la base de datos, recuperando la información más relevante según la similitud semántica. Todo eso gracias al método *similarity_search* de *Chroma*.

Este enfoque permite una gestión eficiente y flexible del conocimiento, adaptándose a la naturaleza dinámica y heterogénea de la información turística.

2.5 Procesamiento de Lenguaje Natural (NLP)

El sistema GPTur incorpora un módulo de procesamiento de lenguaje natural (NLP) para analizar y comprender las consultas de los usuarios, así como para procesar y organizar la información turística recopilada. Las principales tareas de NLP implementadas son:

– *Preprocesamiento de texto:* Se realiza limpieza, normalización y lematización de las consultas y documentos, eliminando palabras vacías y signos de puntuación, utilizando el modelo *es_core_news_md* de *spacy*.
– *Extracción de entidades:* Se identifican entidades nombradas relevantes (como ciudades, atracciones y organizaciones) en las consultas y documentos, facilitando la recuperación precisa de información.
– *Análisis de sentimiento:* Se implementa un análisis de sentimiento simple para detectar la polaridad de las consultas, lo que puede influir en la generación de respuestas más adecuadas.
– *Extracción de palabras clave:* Se extraen los términos más relevantes de cada consulta o documento, mejorando la indexación y recuperación.
– *Fragmentación de texto:* Los textos largos se dividen en fragmentos coherentes para optimizar el procesamiento y la generación de embeddings.

Estas tareas permiten que el sistema interprete correctamente las necesidades del usuario y recupere la información más relevante, integrando el procesamiento de lenguaje natural en la lógica central de los agentes inteligentes.

3 Validación y Experimentación

Esta sección fue un elemento esencial en la etapa de desarrollo y, en muchos casos, influyó en la toma de decisiones. Se realizaron varios experimentos, que incluyen valoraciones de la respuesta del sistema y evaluación de los algoritmos de metaheurísticas.

3.1 Procedimiento de Bootstrapping y Número de Iteraciones

En el procedimiento de bootstrapping se generan, con reemplazamiento, B muestras de tamaño n a partir de la muestra original, calculándose de cada una el estimador $\bar{X}^*$. El error estándar asociado a la distribución de estos estimadores se expresa como:

$$\mathrm{SE}_B = \frac{\sigma}{\sqrt{B}}. \tag{1}$$

Inicialmente se había planteado un umbral del 5% de σ para SE_B lo que implicaba:

$$\mathrm{SE}_B \leq 0.05\,\sigma. \tag{2}$$

Despejando en la inecuación se obtiene

$$\frac{\sigma}{\sqrt{B}} \leq 0.05\,\sigma \quad \Longrightarrow \quad \sqrt{B} \geq 20 \quad \Longrightarrow \quad B \geq 400.$$

En la práctica se utiliza $B = 1000$ para obtener una estabilidad robusta, ya que:

$$\frac{1}{\sqrt{1000}} \approx 0.0316,$$

lo que implica que $\mathrm{SE}_B \approx 0.0316\,\sigma$, cumpliéndose de forma estricta la condición.

3.2 Respuesta del Sistema

Durante las distintas etapas de desarrollo, fue importante evaluar el comportamiento de la aplicación. Para estos experimentos se utilizó un modelo de lenguaje para generar 30 preguntas relacionadas con el corpus y obtener las respuestas de GPTur automáticamente. Luego, se creó una cosulta para un modelo de lenguaje pidiendo una evaluación entre 0 y 5 puntos de la respuesta del sistema en cuanto a:

- Precisión de la información
- Coherencia y relevancia
- Cobertura del tema
- Calidad de las recomendaciones
- Ajuste a la pregunta realizada

Después de obtener los resultados de la muestra, se hacía necesario evaluar muestras mucho más grandes, pero eso requería demasiado cómputo, así que como alternativa se utilizó la técnica conocida como *bootstrapping*, pudiendo estudiar muestras de tamaño 300 con 1000 simulaciones. A continuación se muestran los resultados obtenidos en las distintas etapas.

Sistema con Agentes Guía y Planificador Solamente: La Tabla Tab. 1 resume las métricas clave obtenidas:

Table 1. Resultados del análisis de bootstrapping (IC 95%)

Métrica	Tamaño=30	Tamaño=300
Puntuación Media	3.7	3.7013...
Desviación Estándar	0.4582...	0.0261...

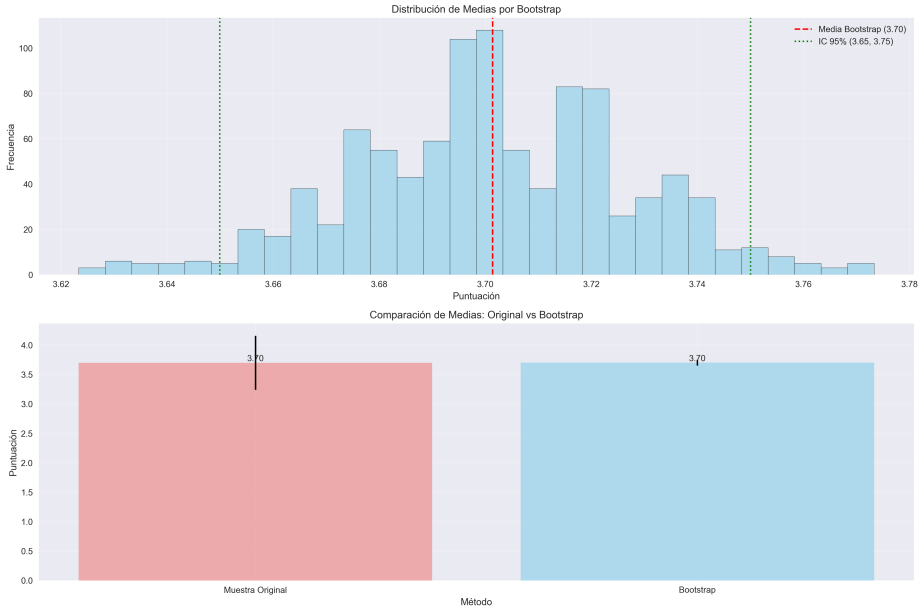

Fig. 1. Distribución bootstrap de la puntuación media ($N = 1000$). La línea discontinua roja indica la media bootstrap (3.70)

Remark 1. Se obtuvo un intervalo de confianza de $[3.6499..., 3.75]$, lo que indica unos resultados aceptables. La Figura Fig. 1 muestra la distribución bootstrap de la puntuación media y la figura Fig. 2 muestra un mapa de calor con las puntuaciones otorgadas a las respuestas del sistema a las 30 preguntas iniciales.

Sistema con Agentes Especializados: La Tabla Tab. 2 resume las métricas clave obtenidas:

Fig. 2. Evaluaciones obtenidas para la respuesta a cada pregunta en la muestra más pequeña (tamaño 30)

Table 2. Resultados del análisis de bootstrapping (IC 95%)

Métrica	**Tamaño=30**	**Tamaño=300**
Puntuación Media	4.1333...	4.1329...
Desviación Estándar	0.6182...	0.0349...

Remark 2. Se obtuvo un intervalo de confianza de $[4.0666..., 4.1966...]$, lo que indica unos resultados bastante buenos. La Figura Fig. 3 muestra la distribución bootstrap de la puntuación media y la figura Fig. 4 muestra un mapa de calor con las puntuaciones otorgadas a las respuestas del sistema a las 30 preguntas iniciales.

En resumen, se puede observar que al añadir agentes especializados al sistema, mejoró la respuesta del modelo, lo cual demostró que la decisión tomada estuvo acertada.

4 Conclusiones

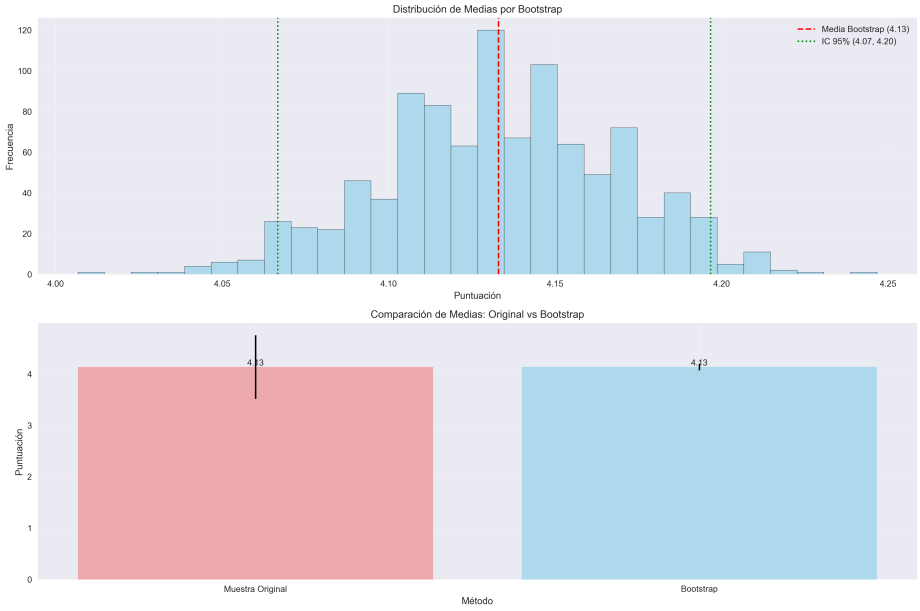

Fig. 3. Distribución bootstrap de la puntuación media ($N = 1000$). La línea discontinua roja indica la media bootstrap (4.13)

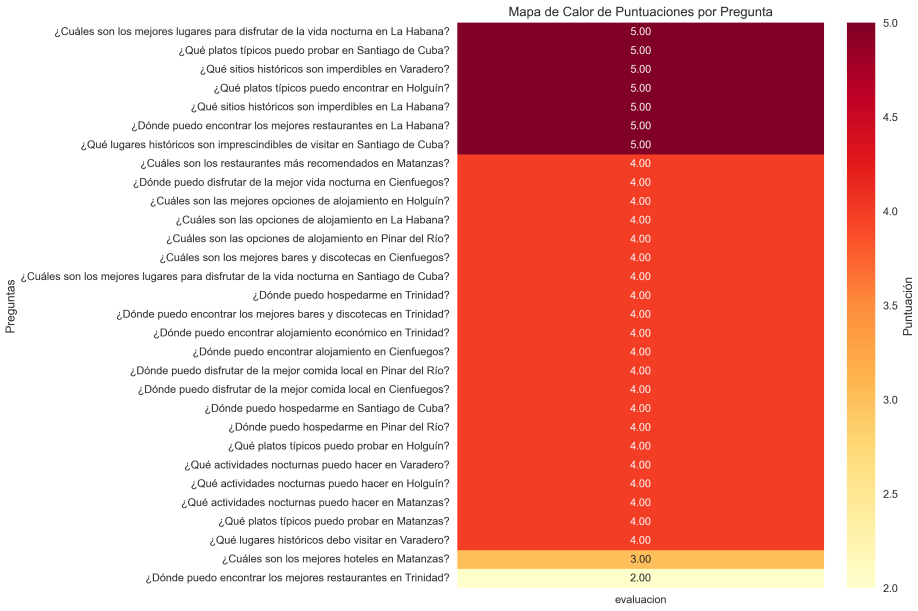

Fig. 4. Evaluaciones obtenidas para la respuesta a cada pregunta en la muestra más pequeña (tamaño 30)