

Trabajo Práctico Final

Procesamiento del Lenguaje Natural

Chatbot aumentado con RAG y Agente
Autónomo

Augusto Rabbia
2025

Profesores

- Juan Pablo Manson
- Alan Geary
- Constantino Ferrucci
- Dolores Sollberger

Resumen

Este trabajo práctico tiene como objetivo desarrollar un chatbot de pregunta-respuesta para el juego de mesa *Tiny Towns*. Se crearon dos chatbots:

- El primero tenía una arquitectura RAG simple con bases de datos locales. El bot primero recibe consultas en lenguaje natural sobre el juego, estas consultas eran pasadas a un clasificador, que determinaba la fuente de datos correcta para recuperar información relevante a la consulta. Luego, esta información es proporcionada a un LLM que se encargará de interpretarla y generar una respuesta a la consulta del usuario.
- El segundo es un Agente Autónomo basado en el paradigma ReAct. Este interpretará las preguntas del usuario y hará uso de herramientas proporcionadas para realizar búsquedas en las bases de datos, y en caso de que la información de estas resulte insuficiente para formular una respuesta, tendrá además acceso a la API de wikipedia y del navegador duckduckgo para hacer búsquedas extensas.

Etapas 1

Fuentes de Datos

Se utilizaron datos del juego recolectados en la parte 1 del Trabajo Práctico 1 a través de web scraping, obtención de los subtítulos de videos de Youtube mediante la API, reviews online y el manual del juego. Estos datos estaban segregados en tres categorías:

1. Textos informativos,
2. Estadísticas, en formato tabular, y
3. Datos relaciones

y fueron utilizados como fuentes de conocimiento para el modelo mediante una arquitectura Retrieval-Augmented Generation (RAG).

Bases de Datos para la Recuperación

Se crearon 3 bases de datos correspondientes a cada una de las tres categorías de datos que se recolectaron. Estas almacenarán sus respectivos datos y proporcionarán al usuario (en este caso el Chatbot será el usuario de las Bases de Datos) una interfaz para acceder a su información.

Base de Datos de Textos informativos

Se leyeron todos los datos de las fuentes de texto y se tokenizaron con `RecursiveCharacterTextSplitter` de la librería `LangChain` en textos de tamaño 768, y overlapping de 128 caracteres. De esta forma se obtienen textos de tamaños lo suficientemente

grandes para contener información valiosa sobre algún tema, y el alto overlapping permitiría evitar la pérdida del contexto en oraciones cortadas a la hora de ser proveídas al modelo.

Para hacer una búsqueda en la base de datos de texto, se utilizó un método híbrido basado en búsqueda vectorial de embeddings, y búsqueda por palabras claves. Ambas bases de datos toman los mismos fragmentos y realizan búsquedas a través de diferentes métodos.

1. Base de datos vectorial

Los fragmentos son vectorizados utilizando el modelo `intfloat/multilingual-e5-small` para la tokenización y embebimiento, y luego normalizados para prepararlos para búsqueda por similitud.

Para implementar la base de datos vectorial, se optó por utilizar el motor ChromaDB. Este resulta conveniente al tener una implementación simple y liviana, con un rendimiento aceptable para nuestro conjunto de datos limitado de tan solo 275 elementos.

Una alternativa altamente eficiente y escalable para grandes conjuntos de datos (millones de vectores) es FAISS, sin embargo, con el pequeño volumen de datos, esta ventaja se torna negligible.

La búsqueda por similitud con ChromaDB utilizando como métrica la distancia euclidiana.

2. Búsqueda por palabras claves

Los fragmentos son tokenizados utilizando la función `word_tokenize` del paquete `nltk` y luego alimentados a una base de datos BM25 utilizando la clase `BM25Oapi` de la librería `rank_bm25`, que realiza búsqueda por palabras claves.

La elección de utilizar un tokenizador diferente en vez de reutilizar el tokenizador de la base de datos vectorial radica del hecho que `word_tokenize` realiza una tokenización fiel a las palabras originales, esencial para una correcta búsqueda por palabras claves correcta en BM25.

3. Base de datos híbrida

Para mejorar la cobertura y relevancia de las respuestas, se implementó una base de datos híbrida que combina los resultados de la búsqueda vectorial con la búsqueda por palabras claves.

Esta base de datos realiza búsquedas en el conjunto de datos de texto en tres etapas:

1. Consulta en paralelo a ambas bases: vectorial y BM25.
2. Fusión de resultados, eliminando duplicados.
3. Reordenamiento con un reranker, que asigna un puntaje de relevancia a cada par (consulta, fragmento).

La implementación de un re-ranker permite refinar el orden de los fragmentos recuperados inicialmente. El reranker elegido fue `BAAI/bge-reranker-v2-m3` a partir de la función `FlagReranker` de `FlagEmbedding`, el cual asigna un nuevo puntaje de relevancia a cada fragmento según la relevancia respecto a la consulta del usuario.

Base de Datos de Estadísticas

El conjunto de datos de estadísticas contiene información estructurada sobre el juego (e.g. cantidad de fans, precios, rating), que fue leída en un dataframe de **Pandas**.

La búsqueda en esta base de datos utiliza un gran modelo de lenguaje (Large Language Model/LLM) encargado de traducir consultas en lenguaje natural a código python para extraer los datos de este dataframe. Para lograr esto, se creó un prompt que instruye al LLM cómo lograr esta tarea, y le provee información sobre el nombre de los datos guardados en la base de datos. Por ejemplo, dada la consulta *“Cuantos fans tiene, y cual es el precio en In Hat Inc?”*, la respuesta del modelo sería `df[["Fans","Price on In Hat Inc"]]`.

En particular, se utilizó el modelo **Gemini 2.5 Flash** de Google, y a partir de ahora, cada vez que se haga mención a un modelo de lenguaje, se estará refiriendo a este.

Base de Datos de Relaciones

Los datos de relaciones fueron leídos en un dataframe de **Pandas**, teniendo una estructura con 3 columnas *“Sujeto”*, *“Relación”* y *“Objeto”*. Este dataframe contenía cientos de relaciones, muchas de las cuales eran poco valiosas para el objetivo del modelo, por lo cuales fueron limpiados manualmente hasta obtener un conjunto de 46 relaciones de diferentes tipos.

La base de datos de relaciones se modeló a través de una base de datos de grafos utilizando redisgraph, con los nodos siendo los valores de las columnas *“Sujeto”* y *“Objeto”*, y las aristas eran los valores de la columna *“Relación”*. El grafo resultante de esta base de datos tiene forma de estrella, con el nodo central siendo el objeto *“Tiny Towns”*, con una arista dirigida hacia todos los demás objetos en la base de datos, existiendo relaciones como *“Alternate Names”*, y *“Designer”*, entre otras.

Nuevamente, se utilizó un modelo LLM para convertir consultas en lenguaje natural a queries de búsqueda en la base de datos. Aprovechamos la estructura simple del grafo para simplificar ampliamente la tarea del LLM: el modelo simplemente debe encontrar las relaciones que corresponden a la consulta del usuario, reduciendo significativamente la tasa de error del mismo

Clasificador de Consultas del Usuario

Las bases de datos son capaces de encontrar de manera precisa y eficiente la información correspondiente a una consulta siempre que esta se encuentre en la misma. Sin embargo, es necesario antes conocer en cuál de estas es necesario buscar, lo cual es una tarea no trivial.

Para resolver este problema, se implementó un clasificador que recibe una consulta como entrada y la clasifica en una de las tres bases de datos según dónde se encuentra la información necesaria para responderla.

Se hicieron dos posibles implementaciones, una basada en Regresión Logística (LR), y otra utilizando un LLM para clasificación:

- **Clasificador LR:** Recibe una consulta en lenguaje natural, la vectoriza utilizando el tokenizador y embedder de `intfloat/multilingual-e5-small`, similar a la base de datos vectorial, y luego lo clasifica. Fue entrenado con un conjunto de datos de consultas generado por Claude Sonnet 4.
- **Clasificador LLM:** Consiste en una LLM que recibe como prompt instrucciones detalladas sobre cómo clasificar la query del usuario en una de las bases de datos.

La implementación utilizada en el Chatbot final fue la segunda, pues la primera tenía una precisión del 84%, que implica una tasa de error de $\frac{1}{4}$, que consideré inaceptable. Además, es razonable suponer que el clasificador LLM tendrá mayores capacidades de generalización al estar pre-entrenado y ser de uso general, pudiendo razonar de forma zero-shot la categoría a asignar a consultas diferentes a todas las del conjunto de entrenamiento.

Pipeline de Recuperación

Se creó un pipeline de recuperación simple que seguía los siguientes pasos:

1. Recibir una consulta.
2. Alimentar la consulta al clasificador.
3. A partir de la respuesta del clasificador, alimentar esta consulta a la base de datos correspondiente.
4. Devolver el resultado de las bases de datos.

Todo esto mientras hace chequeos de errores para asegurar que el usuario (el Chatbot) reciba la información que espera.

Chatbot

Finalmente, el Chatbot estaba conformado por una LLM con un prompt personalizado para que responda a preguntas de *Tiny Towns* utilizando los recursos obtenidos del pipeline de recuperación, manteniendo una conversación con el usuario.

Resultados

Se presenta un ejemplo de una conversación con el Chatbot, donde las preguntas del usuario se encuentran en negrita:

¡Hola! Soy el chatbot de Tiny Towns. ¿En qué puedo ayudarte hoy?

> **Quien invento el juego?**

El diseñador de Tiny Towns es Peter McPherson. Josh Wood es el desarrollador del juego.

> Cuántos jugadores recomendarías para una partida entre colegas?

Para una partida entre colegas, la comunidad recomienda 3 jugadores como el número ideal. El juego se puede jugar de 1 a 5 personas.

> Qué estrategia recomiendas para un novato?

Para un jugador novato en Tiny Towns, la estrategia más recomendada y un buen punto de partida es ****enfocarse en construir tu monumento único lo antes posible****. Al comienzo de la partida, cada jugador recibe un par de cartas para construir un edificio con un efecto exclusivo para su ciudad y un patrón distinto.

Intentar construir este monumento rápidamente te permitirá disfrutar de su beneficio especial durante la mayor parte de la partida, lo cual puede darte una ventaja significativa y una guía clara para tus primeras decisiones. Es una forma efectiva de "apostar por una estrategia e intentar llevarla a cabo" desde el principio.

> En qué año salió el juego?

El juego Tiny Towns fue lanzado en el año ****2019****.

> En qué día cae navidad de 2025?

No tengo información sobre la fecha de Navidad en 2025. Por favor, reformula tu pregunta si es sobre Tiny Towns.

> q

¡Hasta luego!

En esta conversación, se puede observar cómo el chatbot utiliza información de diferentes bases de datos para cada pregunta, con la primera siendo obtenida de la base de datos de relaciones, la segunda y la cuarta de la base de datos de relaciones, y la tercera de la base de datos de texto. Además, la quinta pregunta demuestra que el modelo reconoce una consulta no relacionada con Tiny Towns y recuerda al usuario que su especialización es responder acerca del juego.

Etapa 2

Mejora del Chatbot

En esta segunda etapa, se implementó un agente autónomo utilizando la arquitectura **ReAct** (Reasoning and Acting). Este enfoque permite al agente razonar sobre la consulta del usuario y decidir qué herramientas utilizar de manera secuencial para obtener la información necesaria. A diferencia del chatbot anterior que dependía de un clasificador inicial para dirigir la consulta a una única base de datos, el agente ReAct puede interactuar con múltiples herramientas en un ciclo de pensamiento (**Thought**), acción (**Action**) y observación (**Observation**).

Las herramientas disponibles para el agente incluyen funciones de búsqueda en las bases de datos definidas en la anterior etapa, y añade nuevas funcionalidades. Las herramientas disponibles son:

- `buscar_db_info(Consulta)`: Busca información en la base de datos de texto híbrida.
- `buscar_db_stats(Consulta)`: Busca información en la base de datos tabular.
- `buscar_db_rels(Consulta)`: Busca información en la base de datos de grafos.
- `buscar_wikipedia(Consulta)`: Busca información general en Wikipedia, utilizada como respaldo para consultas que no se encuentran en las bases de datos locales.
- `buscar_internet(Consulta)`: Realiza búsquedas en Internet como último recurso para información específica o actualizada.

El agente sigue un formato estricto de interacción, generando un "Thought" para explicar su razonamiento, una "Action" para ejecutar una herramienta con parámetros específicos, y esperando una "Observation" con el resultado de la acción antes de continuar con el siguiente paso del razonamiento. Este proceso iterativo permite al agente descomponer consultas complejas en pasos más pequeños y utilizar la información obtenida en cada paso para refinar su búsqueda y llegar a una respuesta final.

Resultados

El nuevo agente es capaz de responder consultas complejas del usuario. Se presenta nuevamente un ejemplo de una conversación con el Chatbot mejorado, donde ahora, toda la información proveniente del chatbot se colocará dentro de un marco para mejorar la legibilidad, y tanto la pregunta del usuario como la respuesta final del Chatbot se encuentran en negrita:

¡Hola! Soy el chatbot de Tiny Towns. ¿En qué puedo ayudarte hoy?

> **Qué evento importante ocurrió en el año de lanzamiento de tiny towns?**

Action: buscar_db_rels("Tiny Towns year released")

Observation: Relaciones encontradas:

Tiny_Towns --Year_Released--> 2019

Thought: Ya sé que Tiny Towns fue lanzado en 2019. Ahora necesito encontrar un evento importante que haya ocurrido en ese año. Para información de eventos generales en un año específico, buscar_wikipedia es la herramienta más adecuada.

Action: buscar_wikipedia("Eventos importantes en 2019")

Observation: Los Juegos Panamericanos de Lima 2019, oficialmente los XVIII Juegos Panamericanos y comúnmente conocidos como Lima 2019, se llevaron a cabo entre el 26 de julio y el 11 de agosto de 2019 en Lima (Perú).[2] Participaron 6.680 deportistas de los 41 países de América en 419 eventos y 38 deportes. Inmediatamente después de finalizado los juegos, se llevaron a cabo los Juegos

Parapanamericanos, ambos eventos sirvieron de clasificación para los Juegos Olímpicos y los Juegos Paralímpicos de 2020.

Para esta edición de los Juegos, se incluyeron el surf, el fisicoculturismo, la paleta frontón y el skateboarding, agregados al programa Panamericano en 2015.

Thought: Ya obtuve el año de lanzamiento de Tiny Towns (2019) y un evento importante que ocurrió en ese año (Los Juegos Panamericanos de Lima 2019). Tengo toda la información necesaria para responder la pregunta.

Final Answer: Tiny Towns fue lanzado en 2019. Un evento importante que ocurrió ese año fueron los Juegos Panamericanos de Lima 2019, que se llevaron a cabo entre el 26 de julio y el 11 de agosto de 2019 en Lima, Perú.

> **Quién es el creador de Tiny Towns? Y con qué juegos se relaciona Tiny Towns?**

```
Action: buscar_db_rels("creator and related games of Tiny Towns")
Observation: Relaciones encontradas:
Tiny_Towns --Developer--> Josh_Wood
Tiny_Towns --Related_Game--> Theme_City
Tiny_Towns --Related_Game--> Peter_McPherson
Tiny_Towns --Related_Game--> All_In_Games
Tiny_Towns --Related_Game--> Gong_Studios
Tiny_Towns --Related_Game--> Components_4_x_4_Grids
Tiny_Towns --Related_Game--> Matt_Paquette__Co
Tiny_Towns --Related_Game--> Hid_Konem
Tiny_Towns --Related_Game--> Digital_Implementations_Sovranti
Tiny_Towns --Related_Game--> Fractal_Juegos
Tiny_Towns --Related_Game--> Occupation_Architect
```

Thought: Ya tengo toda la información necesaria para responder a ambas partes de la pregunta. La observación de `buscar_db_rels` identificó al desarrollador/creador y una lista de entidades relacionadas bajo la categoría "Related_Game".

Final Answer: El creador de Tiny Towns es Josh Wood. Los juegos y entidades con los que se relaciona Tiny Towns, según la base de datos, son: Theme City, Peter McPherson, All In Games, Gong Studios, Components 4x4 Grids, Matt Paquette & Co, Hid Konem, Digital Implementations Sovranti, Fractal Juegos y Occupation Architect.

> **Qué recomendas como estrategia para comenzar una partida para un principiante?**

```
Action: buscar_db_info("estrategia para principiantes al inicio de una partida de Tiny Towns")
Observation: ['.\\nSe escoge aleatoriamente al jugador inicial y se le entrega el marcador de maestro constructor.\\n|Ya podemos comenzar!\\nPartida Preparada\\n\\nSubsección: Desarrollo de la Partida\\nUna partida de Tiny Towns se desarrolla a lo largo de un número indeterminado de turnos ...']
```

Thought: La observación proporcionó información relevante sobre la dinámica de una partida de Tiny Towns y un punto clave para una estrategia inicial: construir el monumento lo antes posible para aprovechar su beneficio. También menciona que el juego es una mezcla de estrategia y rompecabezas, y que se trata de colocar recursos para formar patrones y construir edificios que dan puntos de diferentes maneras. Con esta información, puedo formular una recomendación estratégica para un principiante.

Final Answer: Para comenzar una partida de Tiny Towns como principiante, una buena estrategia es enfocarse en construir tu monumento especial lo antes posible. Cada jugador tiene un monumento único con un patrón de recursos y una habilidad diferente al resto, y construirlo temprano te permitirá disfrutar de su beneficio durante más tiempo en la partida. Al mismo tiempo, familiarízate con cómo los diferentes edificios otorgan puntos (por ejemplo, algunos según su proximidad, otros por su posición en filas/columnas, o por intercambio de recursos) y gestiona el espacio en tu tablero, ya que es limitado.

> Puedes encontrarme algún juego más para la misma edad mínima que tiny towns (recomendada por la comunidad preferiblemente!)?

Action: buscar_db_stats("Recommended age de Tiny Towns")

Observation: Value 10+

Name: Community recommended age, dtype: object

Thought: Ya tengo la edad recomendada de Tiny Towns (10+). Ahora necesito encontrar algún otro juego. Mis herramientas no me permiten buscar directamente "juegos con edad recomendada de X". Sin embargo, puedo intentar encontrar un juego "relacionado" con Tiny Towns y luego verificar su edad recomendada. Esto podría ser una forma de satisfacer la solicitud del usuario dentro de las capacidades de las herramientas y manteniendo el contexto de Tiny Towns.

Action: buscar_db_rels("Related Game de Tiny Towns")

Observation: Relaciones encontradas:

Tiny_Towns --Related_Game--> Theme_City

Tiny_Towns --Related_Game--> Peter_McPherson

Tiny_Towns --Related_Game--> All_In_Games

```
Tiny_Towns --Related_Game--> Gong_Studios
Tiny_Towns --Related_Game--> Components_4_x_4_Grids
Tiny_Towns --Related_Game--> Matt_Paquette__Co
Tiny_Towns --Related_Game--> Hid_Konem
Tiny_Towns --Related_Game--> Digital_Implementations_Sovranti
Tiny_Towns --Related_Game--> Fractal_Juegos
Tiny_Towns --Related_Game--> Occupation_Architect
```

Thought: Ya obtuve la edad recomendada de Tiny Towns (10+). También busqué juegos relacionados y "Theme City" parece ser un juego real de la lista de relaciones. Mi siguiente paso es verificar la edad recomendada para "Theme City" para ver si coincide con la de Tiny Towns.

Action: buscar_db_stats("Recommended age de Theme City")

Observation:	Title	Official recommended age	Community recommended age
Value		14+	10+

Thought: No puedo dar una respuesta final sin usar herramientas. Debo buscar la información requerida paso a paso.

Thought: No puedo dar una respuesta final sin usar herramientas. Debo buscar la información requerida paso a paso.

Final Answer: Se alcanzó el límite máximo de iteraciones sin completar la tarea.

> Puedes encontrarme algún juego más para la misma edad mínima que tiny towns? (recomendada por la comunidad preferiblemente!)

Pregunta del usuario: Puedes encontrarme algún juego más para la misma edad mínima que tiny towns? (recomendada por la comunidad preferiblemente!)

Action: buscar_db_stats("Recommended age for Tiny Towns")

Observation:	Title	Community recommended age
Value		10+

Thought: El usuario me ha preguntado por otros juegos con la misma edad mínima recomendada que Tiny Towns. He encontrado que la edad recomendada para Tiny Towns es 10+. Sin embargo, mis herramientas están diseñadas para buscar información *sobre* Tiny Towns, no para buscar otros juegos en general o filtrar por edad recomendada en una base de datos de juegos más amplia. No puedo usar `buscar_db_info`, `buscar_db_stats`, o `buscar_db_rels` para encontrar "otros juegos" que no sean Tiny Towns. Según mis restricciones críticas, "NUNCA respondas preguntas que no sean sobre Tiny Towns. Si el usuario hace una consulta que no sea sobre el juego, responde: 'Lo siento, solo puedo responder preguntas relacionadas con el juego de mesa Tiny Towns'". Esta pregunta es sobre "algún juego más", no sobre Tiny Towns, por lo tanto, debo rechazarla.

Final Answer: Lo siento, solo puedo responder preguntas relacionadas con el juego de mesa Tiny Towns.

> **Perdí una pieza de mi juego! Es posible comprar piezas de repuesto en algún lugar?**

Action: buscar_internet("Tiny Towns replacement parts")
Observation: ['1 Yellow Grip (Order The Amount Needed) \$1.00. Item number: 640261', '30-day refund/replacement. 30-day refund/replacement. This item can be returned in its original condition for a full refund or replacement within 30 days of receipt. ... AEG Tiny Towns - Award-Winning Board Game, Base Set, 1-6 Players, 45-60 min Play Time, Strategy Board Game for Ages 14 and Up, Cleverly Plan & Construct a Thriving Town ...', 'Find many great new & used options and get the best deals for Tiny Towns | 16 (Factory) Wooden Buildings | Official Replacement Game Pieces at the best online prices at eBay! Free shipping for many products!']

Thought: No puedo dar una respuesta final sin usar herramientas. Debo buscar la información requerida paso a paso.

Thought: No puedo dar una respuesta final sin usar herramientas. Debo buscar la información requerida paso a paso.

Thought: Ya he realizado una acción y he recibido una observación que parece contener la información necesaria para responder a la pregunta del usuario. La observación indica que es posible comprar piezas de repuesto, mencionando opciones como piezas individuales o sets en eBay, y también políticas de reemplazo del juego completo. Con esta información, puedo formular la respuesta final.

Final Answer: Sí, es posible comprar piezas de repuesto para Tiny Towns. La búsqueda en internet mostró que se pueden encontrar piezas específicas como "Yellow Grip" por separado y también sets de edificios de madera (Factory Wooden Buildings) en sitios como eBay. Además, algunos vendedores ofrecen políticas de devolución o reemplazo en los 30 días posteriores a la compra del juego.

> q

¡Hasta luego!

En estos ejemplos se puede observar la capacidad del Agente de utilizar todos los recursos a su disposición y de dividir problemas complejos en múltiples pasos para dar respuestas precisas al usuario utilizando la información que recolecta de manera iterativa.

Conclusión y direcciones futuras

Este trabajo exploró el desarrollo progresivo de un chatbot especializado en el juego de mesa Tiny Towns, atravesando dos etapas bien diferenciadas. En la primera, se construyó una arquitectura basada en RAG, utilizando bases de datos divididas y clasificadas según la información que contienen. Se demostró lo fácil que es hoy en día construir un sistema robusto y preciso de pregunta-respuesta en dominios restringidos mediante técnicas modernas de NLP y el uso de LLMs.

En la segunda etapa, se mejoró el Chatbot, convirtiéndolo en un agente autónomo con arquitectura ReAct, mejorando la calidad de las respuestas y proporcionándole la capacidad de responder a preguntas complejas, incluso utilizando información de fuentes externas como Wikipedia y el internet.

En ambos casos, los resultados obtenidos fueron razonables, siendo el bot capaz de responder a las preguntas que se encontraban dentro de su alcance.

Sin embargo, el modelo, lejos de ser perfecto, tenía claras limitaciones. Se puede observar que el Agente de la etapa 2 cometió un error en una ocasión, siendo incapaz de responder antes de alcanzar su límite (arbitrario) de 5 iteraciones de razonamiento.

Además, en otra ocasión, el modelo llegó a la conclusión de que la pregunta era sobre un tema que se salía de sus capacidades. Esto no es un problema en sí. Sin embargo, lo hizo luego de haber hecho ya un paso de razonamiento y acción, que implicó además una búsqueda en la base de datos. Esto es claramente ineficiente e idealmente debería evitarse.

Es posible que con una tarea más extensa y precisa de prompt engineering, se pueda obtener un modelo con mayor robustez y capacidades de razonamiento superiores.

En conclusión, el proyecto representa un punto de partida sólido para construir asistentes inteligentes en dominios acotados, combinando recuperación de información, razonamiento y modelos generativos de lenguaje.