Trabajo de investigación: Inteligencia Artificial aplicada a Nvidia NeMo y Google Colab

Autor: Sergi Funes Ortiz.
Curso: 23/24-24/25
Escuela: Colegio Sant Josep Obrer.
Tutor: Artur Arroyo i Pascual.

Abstract

This research explores the application of Artificial Intelligence (AI) through Nvidia NeMo and Google Colab to create a functional AI system. AI refers to the ability of machines to perform tasks that typically require human intelligence, such as learning, reasoning, and natural language processing. Motivated by the rapid advancements in AI technologies, the study aims to understand its inner workings and develop a system capable of transcribing audio to text, translating Spanish text into English, and generating English audio from text inputs.

The research is divided into theoretical and practical components. The theoretical part involves studying AI fundamentals and understanding Nvidia NeMo and Google Colab, focusing on their roles in AI development. The practical part involves coding in Python, leveraging libraries, and utilizing Google Colab as the primary environment for creating and training the AI system. Nvidia NeMo is used for speech and text processing tasks, while Colab provides computational resources like GPUs for efficient model training.

The outcomes highlight the capabilities of combining Nvidia NeMo and Google Colab to develop AI models for speech-to-text and text-to-speech tasks.

Aquesta investigació explora l'aplicació de la intel·ligència artificial (IA) a través de Nvidia NeMo i Google Colab per crear un sistema d'IA funcional. La IA es refereix a la capacitat de les màquines per realitzar tasques que normalment requereixen intel·ligència humana, com ara l'aprenentatge, el raonament i el processament del llenguatge natural. Motivat pels ràpids avenços de les tecnologies d'IA, l'estudi pretén entendre el seu funcionament intern i desenvolupar un sistema capaç de transcriure àudio a text, traduir text espanyol a anglès i generar àudio en anglès a partir de les entrades de text.

La recerca es divideix en components teòrics i pràctics. La part teòrica consisteix a estudiar els fonaments de la IA i entendre Nvidia NeMo i Google Colab, centrant-se en les seves funcions en el desenvolupament d'IA. La part pràctica consisteix a codificar en Python, aprofitar biblioteques i utilitzar Google Colab com a entorn principal per crear i entrenar el sistema d'IA. Nvidia NeMo s'utilitza per a tasques de processament de veu i text, mentre que Colab proporciona recursos computacionals com les GPU per a una formació eficient de models.

Els resultats destaquen les capacitats de combinar Nvidia NeMo i Google Colab per desenvolupar models d'IA per a tasques de veu a text i de text a veu.

Índice

1.Introducción

El tema seleccionado para este trabajo de investigación es la inteligencia artificial aplicada a Google Colab y Nvidia NeMo. La inteligencia artificial (IA) se refiere a la capacidad de las máquinas para realizar tareas que normalmente requieren de la inteligencia humana. Estas tareas incluyen el aprendizaje, el razonamiento, la resolución de problemas, el reconocimiento de patrones, la comprensión del lenguaje natural y la toma de decisiones. La IA se basa en algoritmos y modelos matemáticos que permiten a las máquinas aprender de datos y experiencias para actuar como un ser humano. La IA tiene aplicaciones en una amplia variedad de campos, como la medicina, la robótica, la educación, la atención al cliente, entre otros.
En este trabajo de investigación, abordaré más en concreto el programa Nvidia NeMo utilizando Google Colab, para poder crear una IA que sea capaz de pasar de audio a texo y de texto a audio.

2. Motivaciones

La razón por la que he escogido este tema es porque en los últimos años se han desarrollado y mejorado cada vez más estas tecnologías y posiblemente se trate de una revolución tecnológica como en su día lo fue la aparición del Internet. Es por eso que me gustaría aprender como funciona una IA por dentro y como aplicarla usando Google Colab y Nvidia NeMo.

2.1 Tipo de trabajo de investigación

a. Tipo de trabajo de investigación:

El tipo de trabajo de investigación es crear una inteligencia artificial propia usando Nvidia NeMo y Google Colab, a través del lenguaje de programación Python y con diversas librerías.

b. Objetivos:

-Investigar sobre la inteligencia artificial y su funcionamiento.
-Conocer que es Nvidia NeMo y Google Colab.
-Crear una IA con Nvidia NeMo y Google Colab capaz de transcribir audio a texto, traducir texto en español al inglés y generar audio en inglés gracias a un texto.

c. Metodología:

Planificación de la parte teórica: Primero investigaré el funcionamiento de una IA y como se entrena. Una vez tengamos un buen repertorio de información y conocimiento sobre la IA en general, procederemos a investigar que es Nvidia NeMo y Google Colab y como con estas dos tecnologías se puede crear una IA.

Método a usar para la parte teórica: Iré recopilando diversas fuentes de información sobre nuestro trabajo para más tarde filtrar y encontrar la información adecuada.

Planificación de la parte práctica: Crearé una IA con Nvidia NeMo y Google Colab usando el lenguaje de programación Python y diferentes librerías de este, la entrenaremos y añadiremos diferentes funcionalidades como transcribir audio a texto, traducir texto en español al inglés y generar audio en inglés gracias a un texto.

Método a usar para la parte práctica: Crearé una hoja de Jupyter Notebook en Google Colab, para crear las primeras funciones de la IA y entrenarla.

Bloque Teórico

3. Inteligencia Artificial

La inteligencia artificial es el conjunto de tecnologías que permite a una computadora realizar una variedad de tareas como lo haría un ser humano, tales como ver, comprender, traducir, analizar datos…

Realmente, la inteligencia artificial no es un ordenador con capacidad de pensar por sí mismo, sino más bien son una serie de algoritmos, una serie de automatizaciones programadas para comportarse de una manera determinada según lo que tú pidas. Los algoritmos son una series de instrucciones ordenadas para resolver un problema. En la inteligencia artificial, el algoritmo supone el paso previo a la escritura de código de la inteligencia artificial que queremos desarrollar.
Por ejemplo, si le pides a una inteligencia artificial a través de un chatbot (por ejemplo, ChatGPT), este usará una serie de algoritmos que entiende lo que le pides y las analiza, para posteriormente utilizar otros algoritmos para buscar y proveer la información o tarea requerida.

La IA tiene aplicaciones en una amplia variedad de campos, como la medicina, la robótica, la educación, la atención al cliente, entre otros.

3.1 Breve historia de la IA.

En 1943, los investigadores estadounidenses Warren McCulloch y Walter Pits presentan el modelo de neuronas artificiales, considerada la primera inteligencia artificial.

En 1950, el matemático británico Alan Turing se hizo la pregunta: ¿Pueden pensar las máquinas? Para eso propuso el Test de Turing, en el que intervienen dos personas y un ordenador. Una persona, el interrogador, se sienta en una sala y escribe las preguntas en un terminal. Tanto el ordenador como la otra persona escriben las respuestas y el interrogador debe de adivinar quien es el ordenador y quien la persona.

A mitades de los años 60, en el 1964, Joseph Wizenbaum desarrolla el primer programa informático de procesamiento de lenguaje natural, Eliza, que simula la conversación con humanos.

Dos años más tarde, Shakey es presentado al mundo, siendo el primer robot móvil de propósito general capaz de razonar sobre sus acciones.

A finales de siglo, en el año 1997, la supercomputadora Deep Blue de IBM vence al campeón mundial de ajedrez Garry Casparov, atrayendo el interés mundial en la inteligencia artificial.

En el 2011, Apple lanza Siri, el primer asistente virtual capaz de hablar con seres humanos, dar información, siendo el antecesor de otros asistentes virtuales. Ese mismo año, el sistema Watson de IBM, capaz de responder preguntas de manera natural, gana el concurso de televisión estadounidense “Jeopardy!”.

En el 2014, el programa Eugene logra pasar el test de Turing al convencer a los jueces de que es un ser humano, demostrando los rápidos avances de la inteligencia artificial.

En el año 2022, OpenAI lanza al público ChatGPT, una aplicación de chatbot de inteligencia artificial capaz de mantener conversaciones fluidas, analizar datos, dar información… El lanzamiento de este chatbot aumento exponencialmente el interés mundial por la inteligencia artificial, haciendo que otras empresas como Google, Amazon o Nvidia se interesen en dedicar recursos y dineros en la investigación de la inteligencia artificial.

3.2 Tipos de IA

Podemos diferenciar cuatro tipos diferentes de inteligencia:

- Sistema que piensan como humanos: se centran en emular la inteligencia humana, tanto su comportamiento como pensamiento.
- Sistemas que actúan como humanos: se enfocan en la emulación de la inteligencia humana, pero solo en términos de comportamiento.
- Sistemas que piensan racionalmente: se centran en la resolución de problemas de manera lógica y racional.
- Sistemas que actúan racionalmente: se enfocan en la toma de decisiones, buscando tomar la mejor decisión según la información que poseen.

Tambiés podemos diferenciar entre IA débil y IA fuerte. La IA débil, que se enfoca en tareas específicas y limitadas (asistentes virtuales, reconocimiento de voz…), y la IA fuerte, que busca alcanzar un nivel de inteligencia como el humano (inteligencias

artificiales capaces de leer, comprender, analizar, expresarse naturalmente). Un ejemplo de inteligencia artificial fuerte sería ChatGPT.

3.3 Tipos de aprendizaje

Una inteligencia artificial, para funcionar correctamente, debe aprender tal y como lo haría un ser humano. Lo podemos hacer mediante aprendizaje automático o aprendizaje profundo.

El aprendizaje automático es la habilidad de que una IA aprenda y mejore de forma autónoma mediante redes neuronales, sin tener que ser programado directamente, sino mediante el ánalisis de grandes cantidades de datos.

El aprendizaje profundo es un método de la inteligencia artificial que enseña a las computadoras a procesar datos de una manera que se inspira en el cerebro humano. Los modelos de aprendizaje profundo son capaces de reconocer patrones complejos en imágenes, textos, sonidos y otros datos, a fin de generar información y predicciones precisas.

4.Google Colab

4.1 Definición Google Col·lab

Google Colab, también conocido como “Colaboratory”, es una herramienta online desarrollada por Google que permite programar y ejecutar código en Python a través de tu navegador web. Es una herramienta esencial para programar inteligencia artificial y el análisis de datos de manera rápida y eficaz y es totalmente gratuita.

4.2 Ventajas Google Colab

- No requiere configuración.
- Acceso a CPUs, GPUs y TPUs sin coste adicional.
- Permite compartir contenido fácilmente.

Google Colab es perfecto para aquellas personas que no dispongan de una CPU, GPU o TPU o que simplemente las suyas no sean lo suficientemente potentes. De esta manera, cualquier persona con conexión a internet puede programar una IA sin necesidad de disponer de un ordenador potente ni de elementos específicos para crear la IA.

Google Colab utiliza Jupyter Notebook. Esta herramienta permite que los usuarios puedan editar y ejecutar documentos como si se tratase de un bloc de notas.

Otra característica clave es la capacidad de Google Colab para proporcionar a los usuarios acceso gratuito a hardware acelerado, como las Unidades de Procesamiento Gráfico (GPU) y las Unidades de Procesamiento Tensorial (TPU). Estos recursos son esenciales para muchas tareas de investigación, especialmente en el campo del aprendizaje automático y la inteligencia artificial, donde el procesamiento paralelo puede acelerar significativamente los tiempos de cálculo.

Los usuarios de Google Colab obtienen acceso gratuito a los tiempos de ejecución de GPU y TPU durante un máximo de 12 horas. El tiempo de ejecución de la GPU consta de una CPU Intel Xeon a 2,20 GHz, 13 GB de RAM, una GPU Tesla K80 y 12 GB de VRAM GDDR5. El tiempo de ejecución de la TPU consta de una CPU Intel Xeon a 2,30 GHz, 13 GBde RAM y una TPU en la nube con 180 teraflops de potencia de cálculo. Con Google Colab Pro, puede encargar más CPU, GPU y TPU durante más de 12 horas.
En Google Colab los cuadernos de Jupyter Notebook creados se pueden compartir a otros usuarios, para así trabajar conjuntamente en proyectos. De esta manera, ahora se pueden crear enlaces compatibles para los archivos de Google Colab que están guardados en Google Drive. Al codificar conjuntamente, todo el cuaderno de Python se actualiza automáticamente a medida que la codificación avanza, dando la misma sensación que al editar un documento en Google Drive.

También permite instalar bibliotecas especiales que no se encuentran integradas dentro de Google Colab, tales como AWS S3, GCP, SQL, MySQL, etc, las cuales no están disponibles en los fragmentos del código. Para instalarlas, todo lo que hay que hacer es añadir un código de una línea con los siguientes prefijos de código:

!pip install (ejemplo: !pip install matplotlib-venn)
!apt-get install (ejemplo: !apt-get -qq install -y libfluidsynth1)

Google Colab no solo permite instalar bibliotecas especiales, sino que también ofrece múltiples bibliotecas preinstaladas para que pueda importar la biblioteca que necesite a través de los fragmentos de código. Las bibliotecas son las siguientes: NumPy, Pandas, Matplotlib, PyTorch, TensorFlow, Keras y más bibliotecas ML.

Uno de los puntos fuertes de Google Colab es la capacidad de almacenar en la nube nuestros cambios en el código. De esta manera, con un dispositivo con conexión a internet y acceso a su cuenta de Google, el usuario puede editar y ejecutar el código desde cualquier lugar donde haya acceso a internet. No solo eso, sino que el almacenamiento en la nube también funciona como copia de seguridad ante cualquier incidente. Además, su integración con la plataforma Github permite importar y exportar archivos de código de cualquier repositorio.

Una gran ventaja de Google Colab es que soporta varias fuentes de datos para sus proyectos de entrenamiento de IA. Por ejemplo, puedes importar datos desde una máquina local, montar Google Drive en una instancia de Colab u obtener datos remotos y clonar repositorios de Github.

Por último, al igual que Google Drive, Google Colab permite ver los cambios en el Jupyter Notebook. Así, en caso de cualquier problema, podemos restaurar una versión antigua sin ningún tipo de problema.

4.3 Diferencias entre Google Colab y Jupyter Notebook

Las diferencias entre Google Colab y Jupyter Notebook son bastante importantes. A diferencia de Jupyter Notebook, Google Colab si permite la visualización instantánea de archivos de código, se pueden compartir los documentos de código, tiene bibliotecas preeinstaladas, permite el alojamiento en la nube y la sincronización de archivos.

Además, Google Colab no necesita instalaciones de software en la máquina local (Jupyter Notebook sí), Google Colab está basado en la nube, por lo tanto, se obtendrá un control de versiones automático. Además, Google Drive guarda el cuaderno Python (en cambio, Jupyter Notebook debe de guardar el cuaderno periódicamente), se hacen copias de seguridad de los cuadernos en Google Colab (Jupyter Notebook no realiza copias de seguridad automáticas) y por útlimo el programador puede enviar sus archivos a cualquier persona, incluso a un cliente, el cual no necesitará instalar ningún software (a diferencia de Jupyter Notebook).

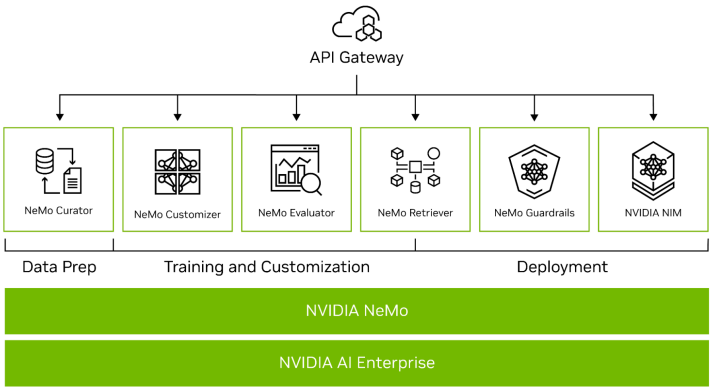
5.Nvidia NeMo

5.1 Definición Nvidia NeMo

Diseñado para el desarrollo empresarial, Nvidia NeMo es una plataforma para crear aplicaciones personalizadas de IA generativa en cualquier lugar. Ofrece un conjunto de microservicios de última generación para permitir un flujo de trabajo completo, desde la automatización del procesamiento de datos distribuidos hasta el entrenamiento de modelos a medida a gran escala. Los modelos creados en Nvidia NeMo se pueden implementar en Nvidia NIM, una serie de microservicios para la implementación de IA generativa. Para las empresas también existe Nvidia IA Enterprise, ofreciendo el tiempo de ejecución más rápido y eficiente a los modelos de IA generativa. Incluye Nvidia NeMo y Nvidia NIM.

5.2 IA Generativa lista para la producción con Nvidia NeMo

Nvidia NeMo simplifica y ayuda a la creación de IA generativa personalizada de nivel empresarial al ofrecer una amplia gama de microservicios.



Estas son las herramientas ofrecidas por Nvidia NeMo para la creación de IA generativa:

- NeMo Curator para la curación de datos acelerada por GPU de conjuntos de datos de entrenamiento de alta calidad.
- NeMo Customizer para simplificar el ajuste fino y la alineación de los LLM.
- NeMo Evaluator para la evaluación automática de la precisión de los LLM.
- NeMo Retriever para conectar modelos personalizados a datos empresariales patentados mediante RAG.
- NeMo Guardrails para proteger las aplicaciones de IA generativa de una organización.

5.3 NeMo Curator

Nvidia NeMo Curator aborda los desafíos de la selección de billones de tokens en conjuntos de datos multilingües. A través de su adaptabilidad esta herramienta le permite manejar sin esfuerzo tareas como la descarga de datos, la extracción de texto, la limpieza, el filtrado, la deduplicación exacta o difusa y la descontaminación de tareas posteriores multilingües. Aprovechando el poder de tecnologías de vanguardia como Dask, RAPIDS cuDF, RAPIDS cuGraph y PyTorch, NeMo Curator puede agilizar los procesos de curación de datos en miles de GPU en red, lo que reduce significativamente los esfuerzos manuales y acelera el flujo de trabajo de desarrollo. Esta tecnologia utiliza la GPU antes de la CPU para la duplicación de datos, ya que se ha demostrado que el uso de la GPU para la duplicación es 26 veces más rápido y 6,5 veces más barato, también aumentando la eficacia.

Existen diferentes desafíos a la hora de entrenar las LLM (modelo de aprendizaje automático entrenado en grandes cantidades de texto para comprender y generar lenguaje humano) con millones de parámetros. Esta intensa tarea requiere una amplia potencia de computación distribuida (computadoras que se pasan mensajes entre sí dentro de la arquitectura de los sistemas distribuidos), clústeres(agrupaciones) de hardware y memoria basados en aceleración, frameworks (estructura) de machine learning (ML) fiables y adaptables y sistemas tolerantes a fallos.

Nvidia NeMo utiliza de forma inteligente los recursos de a GPU y la memoria en todos los nodos, o que se traduce en un aumento considerable de la eficiencia. Al dividir el modelo y los datos de entrenamiento,Nvidia NeMo permite un entrenamiento sin problemas de múltiples nodos y múltiples GPU, reduciendo notablemente el tipo de entrenamiento y mejorando la productividad.

5.4 Nvidia NeMo Aligner

NeMo Aligner es un conjunto de herramientas diseñado para una alineación eficiente de modelos. Admite algoritmos de alineación avanzados, incluidos SteerLM (técnica diseñada para proporcionar a los usuarios un mayor control sobre la salida generada por los modelos de lenguaje), DPO (técnica que se centra en mejorar los modelos de lenguaje basándose en las preferencias directas de los usuarios. En lugar de solo entrenar un modelo utilizando grandes cantidades de texto de manera convencional, DPO utiliza feedback específico y preferencias de los usuarios para guiar el entrenamiento del modelo, haciendo que las respuestas generadas sean más alineadas con lo que los usuarios realmente desean o consideran correcto) y aprendizaje reforzado a partir de retroalimentación humana (RLHF), que ayudan a ajustar los modelos de lenguaje para que sean más seguros, menos dañinos y más beneficiosos. El conjunto de herramientas atiende a una amplia gama de tamaños de modelos y emplea técnicas de paralelismo para optimizar el rendimiento y la eficiencia de los recursos en las tareas de alineación de modelos.

5.5 Modelos de Nvidia AI Foundation

Nvidia Nemo utiliza una amplia de modelos preentrenados para crear sus LLM personalizados, ahorrando tiempo y recursos. Al omitir las fases de recopilación y limpieza de datos para entrenar un LLM, puede centrarse en ajustar el modelo utilizando un conjunto de datos más pequeños y específicos, llegando antes a la solución final. Además, los modelos preentrenados vienen con conocimientos preexistentes, listos para ser personalizados.

Nvidia AI Foundation incluye modelos de la comunidad optimizados para el rendimiento y modelos oficiales de Nvidia de nivel empresarial.

También, dispones de la herramienta Nvidia TensorRT-LLM que optimiza los modelos de Nvidia AI Foundation para la latencia y el rendimiento, ofreciendo la mejor experiencia.

Con el catálogo de APIs de Nvidia, los desarrolladores pueden experimentar los modelos en un navegador o prototipos con puntos de conexión de API alojados en Nvidia. Y una vez el desarrollador esté listo para la implementación automática, los modelos se pueden descargar y ejecutar en cualquier centro de datos, nube o workstation (estación de trabajo) acelerados por GPU, como Google Colab (nube con acceso a recursos de GPU).

5.6 Nvidia NeMo Customizer

Nvidia NeMo Customizer aporta un conjunto de capacidades avanzadas para el machine learning. Una de sus características más destacadas es la compatibilidad con técnicas de ajuste fino y alineación de última generación, lo que permite a los usuarios lograr un rendimiento superior del modelo ajustándolos a sus necesidades específicas. Nvidia NeMo Customizer además aprovecha técnicas avanzadas de paralelismo (que una tarea se divida en subtareas independientes y ejecutadas simultáneamente), que no solo mejoran el rendimiento de entrenamiento, sino que también reducen exponencialmente el tiempo para entrenar modelos complejos.

Esta herramienta utiliza múltiples GPU y múltiples nodos para abordar los mayores desafíos en el campo del deep learning (entrenar a una computadora para que realice tareas como las hacemos los seres humanos, como el reconocimiento del habla, la identificación de imágenes o hacer predicciones).

5.7 Nvidia NeMo Evaluator

A medida que las organizaciones y empresas adaptan cada vez más los LLM para satisfacer sus necesidades, surge la necesidad de evaluar y optimizar los modelos para garantizar la máxima precisión y capacidad de respuesta.

Nvidia NeMo Evaluator simplifica esta tarea a través de capacidades de evaluación comparativa automatizadas. El micoservicio proporciona un diseño abierto y extensible, permitiendo la evaluación de modelos con respectos puntos de referencia académicos y conjuntos de datos personalizados. Nvidia NeMo Evaluator está diseñado para garantizar la eficiencia y la flexibilidad de los LLM que se ejecutan localmente, en la nube o data center.
Nvidia NeMo Evaluator amplía las capacidades de Nvidia NeMo Curator y Nvidia NeMo Customizer.

5.8 Nvidia NeMo Retriever

Nvidia NeMo Retriever es una colección de microservicios que permite la búsqueda de datos empresariales para ofrecer respuestas altamente precisas.
Las tareas a las que pueden ser sometidas son las siguientes, entre otras:

- Procesamiento de grandes volúmenes de documentos en forma de archivos PDF, documentos de Office y otros archivos de texto enriquecido.
- Codificación y almacenamiento de estos documentos para la búsqueda semántica.
- Interactuar con bases de datos relacionales existentes.
- Búsqueda de información relevante para responder preguntas.

Con este utensilio, las organizaciones pueden acceder a la información con la latencia más baja, el rendimiento más alto y la máxima privacidad de datos.

5.9 Nvidia NeMo Guardrails

NeMo Guardrails es un kit de herramientas de código abierto para desarrollar fácilmente sistemas conversacionales LLM seguros y confiables que funcionan con todos los LLM, incluidos ChatGPT de OpenAI y Nvidia NeMo.

Los guardrails (guardarrailes) actúan como restricciones o reglas programables que regulan la interacción entre los usuarios y los LLM. Lo hacen de manera similar a como los guardails físicos (guardarrailes) en las autopistas evitan que los vehículos se desvíen de su curso, estos guardrails digitales están diseñados para monitorear, influir y controlar las interacciones del usuario con la IA. Así, se evita que la IA genere contenido alucinante, tóxico o engañoso y bloqueando comandos maliciosos o acceso no autorizado a aplicaciones de terceros.

Nvidia NeMo Guardrails facilita la codificación e implementación de estas medidas de seguridad. Al integrarse con el ecosistema LLM y admitir frameworks populares, Nvidia NeMo Guardrails garantiza que las aplicaciones de IA generativa permanezcan seguras y alineadas con los valores, políticas y objetivos de la organización.

5.10 Nvidia NIM

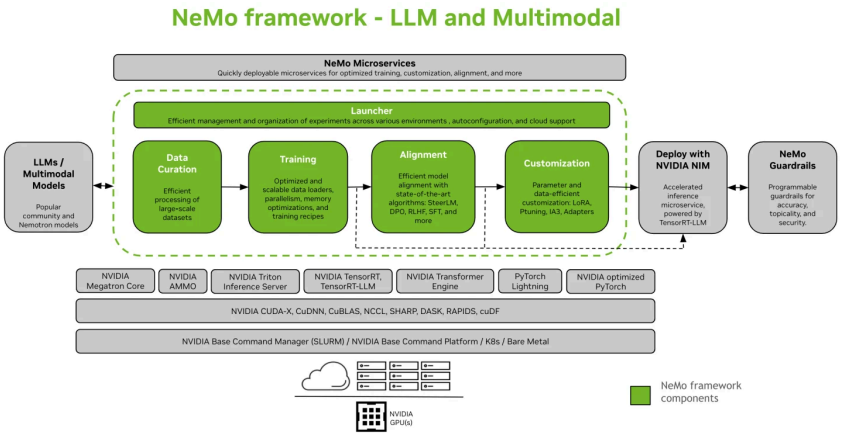
Los microservicios de Nvidia NIM simplifican el camino para implementar modelos de IA generativa para entornos empresariales.
Al aprovechar las APIs estándar de la industria, los desarrolladores pueden crear rápidamente y eficazmente aplicaciones de IA de nivel empresarial con muy pocas líneas de código.

6.Descripción general Nvidia NeMo Framework

Nvidia NeMo framework es una herramienta de IA generativa escalable y nativa en la nube creada para investigadores y desarrolladores que trabajan en modelos de lenguaje grandes, multimodales y IA del habla (reconocimiento automático de voz y texto de voz). Permite a los usuarios crear, personalizar e implementar de manera rápida y eficaz modelos de IA generativa, aprovechando el código y modelos previamente entrenados.

7. Large Language Models and Multimodal

Nvidia NeMo framework provee funcionalidades de desarrollo de modelos de extremo a extremo para LLM y modelos multimodales que se puede utilizar localmente o con un servicio en la nube.



7.1 Curación de datos (Data Curation)

Nvidia NeMo Curator es una librería de Python que consiste en una colección de módulos de minería de datos optimizados para GPU para curar datos en lenguaje natural para entrenar modelos de lenguaje grandes (LLM). Los módulos de NeMo Curator permiten a los investigadores de PNL (programación neurolingüística) extraer texto de alta calidad a escala a partir de datos web masivos sin procesar.

7.2 Entrenamiento y personalización de modelos (Model Training and Customization)

Nvidia NeMo framework incluye todas las herramientas necesarias para entrenar eficazmente y personalizar las LLM y modelos multimodales. Esto incluye configurar el clúster de computación, descargar datos y seleccionar hiperparámetros del modelo. Las configuraciones por defecto para cada modelo y tarea son testeadas

regularmente y cada configuración puede ser modificada para entrenar en nuevos conjuntos de datos o probar nuevos hiperparámetros del modelo. Para la personalización, además del ajuste fino totalmente supervisado (SFT), el framework también soporta una amplia variedad de técnicas de ajuste fino eficiente de parámetros (PEFT), como Ptuning, LoRA, Adapters e IA3, que generalmente pueden alcanzar casi el mismo grado de precisión con una fracción del cospe computacional de las SFT.

7.3 Alineación del modelo (Model Alignment)

Como parte del framework, Nvidia NeMo Aligner es un conjunto de herramientas escalables para una alineación eficiente de modelos. El kit de herramientas admite el ajuste fino supervisado (SFT) y otros algoritmos de alineación de modelos de última generación (SOTA), como SteerLM, DPO, y aprendizaje reforzado a partir de retroalimentación humana (RLHF). Estos algoritmos permiten a los usuarios alinear los modelos de lenguaje para que sean más seguros y útiles.

7.4 Launcher

Nvidia NeMo Launcher optimiza la experiencia con el framework de Nvidia NeMo, ofreciendo una interfaz fácil de usar para el usuario para crear flujos de trabajo de extremo a extremo para una gestión y organización eficiente de experimentos en varios entornos. Construido sobre el framework the Hydra, permite a los usuarios componer y ajustar sin esfuerzo configuraciones básicas utilizando archivos de configuración y argumentos de línea de comandos. Puede comenzar con trabajos de capacitación, personalización o alineación a gran escala localmente (con soporte de un solo nodo), en Nvidia Base Command Manager (Slurm) o proveedores de la nube - AWS, Azure, y Oracle Cloud Infrastructure (OCI) con scripts de inicio sin tener que escribir código.

7.5 Inferencia del modelo (Model inference)

Nvidia NeMo framework se intergra perfectamente con herramientas de implementación de modelos de nivel empresarial utilizando Nvidia NIM, impulsado por Nvidia TensorRT-LLM y Nvidia Triton Inference Server para una inferencia optimizada y escalable.

7.6 Soporte de modelo (Model support)

Los flujos de trabajo de desarrollos de modelos de extremo a extremo para modelos comunitairios populares (Gemma, Starcoder 2, Llama ½, Baichuan 2, Falcon, Mixtral, Mistral, entre otros) y modelos Nvidia Nemotron son compatibles con el

framework de Nvidia NeMo. Esto incluye soporte para entrenamiento previo y ajuste para todos los modelos de lenguaje y modelos multimodales. Las demás características se muestran en las imágenes contiguas.

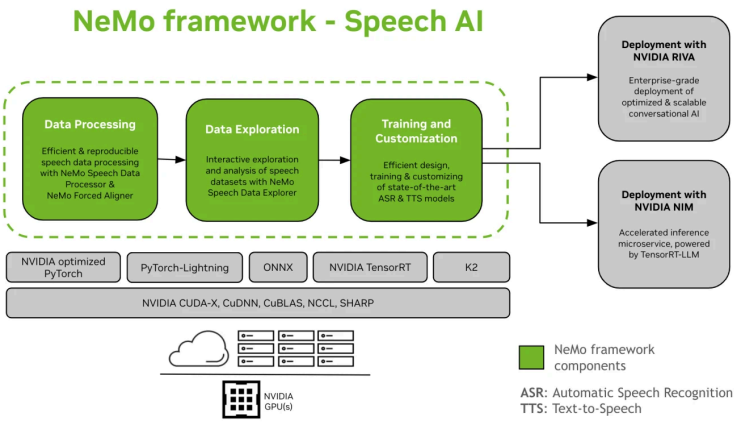
Modelos LLM	PEFT	Alineación	**Verificación de la convergencia de la formación FP8**	TRT/TRTLLM Export	Convertir hacia y desde HF	**Evaluación**
Llama 1/2/3 & Code Llama 2	Sí	Sí	Verificado para algún tamaño	Sí	Ambos	Sí
Gemma & CodeGemma	Sí	Sí	Compatible pero sin verificación	Sí	Solo HF a NeMo	Sí
SpaceGemma (Greiffin)	Sí	No	Sin soporte	No	Ambos	Sí
Baichuan 2	Sí	Sí	Compatible pero sin verificación	Sí	Ambos	Sí
ChatGLM 2/3	Sí	Sí	Compatible pero sin verificación	Sí	Ambos	Sí
Falcon	Sí	Sí	Compatible pero sin verificación	Sí	Ambos	Sí
Mistral 7B	Sí	Sí	Compatible pero sin verificación	Sí	Ambos	Sí
Mistral 8x7B/8x22B	Sí	Sí	Compatible pero sin verificación	No	Ambos	Sí
StarCoder 1/2	Sí	Sí	Compatible pero sin verificación	Sí	Ambos	Sí
T5	Sí	No	Compatible pero sin verificación	No	No	Sí

mT5	Sí	No	Compatible pero sin verificación	No	No	Sí
GPT	Sí	Sí	Verificado para algún tamaño	Sí	No	Sí
BERT	Sí	No	Sin soporte	No	Ambos	No

Modelos Multimodales	PEFT	Alineación	**Verificación de la convergencia de la formación FP8**	TRT/TRTLLM Export	Convertir hacia y desde HF	**Evaluación**
NeVA (Visión-LLM)	Sí	No	Compatible pero sin verificación	Sí	Solo HF a NeMo	Sí
CLIP (VLM)	No	No	Sin soporte	Sí	Solo HF a NeMo	Sí
VIT (VLM)	No	No	Sin soporte	Sí	No	Sí
NSFW (VLM)	NA	No	Sin soporte	Sí	No	NA
SD (Texto a imagen)	Sí	Sí	Verificado para algún tamaño	Sí	Solo HF a NeMo	Sí
Dreambooth (Imagen a Imagen)	Sí	No	Sin soporte	Sí	No	NA
ControlNet (Imagen + Texto a imagen)	NA	No	Sin soporte	Sí	No	NA
InstructPix2Pix (imagen + Texto a imagen)	NA	No	Sin soporte	Sí	No	NA
Imagen (Texto a imagen)	NA	No	Sin soporte	Sí	No	Sí
DreamFusion (Texto a 3D)	NA	No	Sin soporte	Sí	No	NA

8. Speech AI

El desarrollo de modelos de IA conversacional es un proceso complejo que implica definir, construir y entrenar modelos en dominios específicos. Esto a menudo implica múltiples iteraciones para lograr alta precisión, ajustar varias tareas y datos específicos del dominio, garantizar el rendimiento del entrenamiento y preparar modelos para la implementación de inferencia.



El framework de Nvidia Nemo admite el entrenamiento y la personalización de modelos de inteligencia artificial del habla para tareas como el reconocimiento automático de voz (ASR) y la síntesis de texto a voz (TTS), con un camino fluido hacia la implementación de nivel empresarial con Nvidia Riva. Para ayudar aún más a los desarrolladores e investigadores, viene con puntos de control previamente entrenados (SOTA) de última generación y herramientas para el procesamiento de datos de voz reproducibles y exploración y análisis interactivos de conjuntos de datos de voz. Las partes del Nvidia Nemo framework para voz incluyen diversas herramientas que hablaremos a continuación.

8.1 Entrenamiento y personalización (Training and Customization)

El framework de Nvidia NeMo contiene todo lo necesario para entrenar y personalizar modelos de voz (ASR, Speech Classification, Speaker Recognition, Speaker Diarization y TTS) de manera reproducible.

8.2 Modelo entrenado SOTA

Nvidia proporciona documentación y puntos de control previamente entrenados de varios modelos ASR y TTS, así como instrucciones sobre cómo cargarlos.

8.3 Herramientas de voz

Un conjunto de herramientas útiles para desarrollar modelos ASR y TTS, que incluyen:

- NeMo Forced Aligner (NFA) para generar marcas de tiempo de voz a nivel de token, palabra y segmento en audio utilizando los modelos de reconocimiento automático de voz basado en CTC (Clasificación temporal Conexionista o en inglés Connectionist Temporal Classification) de Nvidia NeMo.
- Speech Data Processor (SDP), un conjunto de herramientas para simplificar el procesamiento de datos de voz. Permite a los usuarios representar operaciones de procesamiento de datos en un archivo de configuración, minimizando el código repetitivo y permitiendo la reproducibilidad y la capacidad de compartir.
- Speech Data Explorer (SDE), una aplicación web basada en Dash (biblioteca de Python diseñada para la creación de aplicaciones web interactivas y de análisis de datos) para la exploración y análisis interactivos de conjuntos de voz.
- Dataset Creation Tool para que los modelos ASR que proporciona funcionalidad para alinear archivos de audio largos con las transcripciones correspondientes y dividirlos en fragmentos más cortos que son adecuados para el modelo de reconocimiento automático de voz (ASR).
- Comparision Tool para que los modelos ASR comparen predicciones de diferentes modelos ASR a nivel de precisión de palabras y expresión.
- ASR Evaluator para evaluar el rendimiento de los modelos ASR y otras funciones tales como la detección de actividad de voz.

8.4 Camino hacia la implementación

Los modelos de Nvidia NeMo entrenados o personalizados utilizando el framework específico se pueden optimizar e implementar con Nvidia Riva, que incluye contenedores gráficos de helm diseñados para automatizar los pasos para la implementación con botones.

9. Ventajas de usar Nvidia NeMo

El desarrollo de modelos de aprendizaje profundo para Gen AI (inteligencia artificial generativa) es un proceso complejo que abarca el diseño, la construcción y el entrenamiento de modelos en dominios específicos. Lograr una alta precisión requiere una experimentación extensa, ajustes para diversas tareas y conjuntos de datos de dominios específicos, garantizar un rendimiento de entrenamiento óptimo y preparar modelos para su implementación.

Nvidia NeMo simplifica este complejo panorama de desarrollo a través de su enfoque modular. Introduce módulos neuronales (bloque lógicos de aplicaciones de IA con entradas y salidas tipificadas) que facilitan la construcción perfecta de modelos al encadenar estos bloques en función de tipos neuronales. Esta metodología acelera el desarrollo, mejora la precisión del modelo en datos de dominios específicos y promueve la modularidad, la flexibilidad y la reutilización dentro de los flujos de trabajo de IA.

Para mejorar aún más su utilidad, Nvidia NeMo provee colecciones de módulos diseñados para tareas básicas de reconocimiento de voz, procesando el lenguaje natural y síntesis de voz. Admite el entrenamiento de nuevos modelos o el ajuste de módulos previamente entrenados existentes, aprovechando previamente entrenados para acelerar el proceso de capacitación.

El framework abarca modelos entrenados y optimizados para múltiples idiomas y ofrece tutoriales extensos para el desarrollo de IA conversacional en estos idiomas. El énfasis de Nvidia NeMo en la interoperabilidad con otras herramientas de investigación amplía su aplicabilidad y facilita el uso.

10. Modelos de lenguajes grandes y multimodales (LLM y MM)

Nvidia NeMo se destaca en la capacitación de LLM y MM a gran escala, utilizando optimizaciones de Megatron-LM y Transformer Engine para ofrecer un rendimiento de última generación. Incluye un conjunto integral de funciones para capacitación a gran escala:

- Admite computación multi-GPU y multi-nodo para permitir la escalabilidad.
- Opciones de precisión que incluyen FP32/TF32, FP16, BF16 y TransformerEngine/FP8.
- Estrategias de paralelismo: paralelismo de datos, paralelismo tensorial, paralelismo de canalizaciones, paralelismo de canalizaciones intercaladas, paralelismo de secuencias y paralelismo de contexto, optimizador distribuido y paralelo de datos totalmente compartidos.
- Utilidades optimizadas como Flash Attention, Activation Recomputation y Communication Overlap.
- Puntos de control avanzados a través del formato de puntos de control distribuidos.

11. Información adicional Speech AI

11.1 Aumento de datos

Aumentar los datos de ASR es esencial, pero puede llevar mucho tiempo durante el entrenamiento. Nvidia NeMo aboga por el preprocesamiento de conjuntos de datos fuera de línea para conservar el tiempo de entrenamiento, como se ilustra en un tutorial que cubre técnicas de perturbación de velocidad y aumento de ruido.

11.2 Explorador de datos de voz

Una herramienta basada en Dash para la exploración interactiva de conjuntos de datos ASR/TTS, que proporciona información sobre estadísticas de conjuntos de datos, inspecciones de expresiones y análisis de errores. Las instrucciones de instalación para esta herramienta están disponibles en el repositorio de GitHub de Nvidia NeMo.

11.3 Usando datos formateados Kaldi

Nvidia NeMo admite conjuntos de datos con formato Kaldi, lo que permite el desarrollo de modelos con datos Kaldi existentes sustituyendo AudioToTextDataLayer por KaldiFeatureDataLayer.

11.4 Reconocimiento de comandos de voz

La capacitación especializada para el reconocimiento de comandos de voz se cubre en un cuaderno dedicado de Nvidia NeMo Jupyter, que guía a los usuarios a través del proceso de entrenamiento de un modelo QuartzNet en un conjunto de datos de comandos de voz.

12. Optimizaciones Generales

12.1 Entrenamiento de precisión mixto

Al utilizar Apex AMP de Nvidia, el entrenamiento de precisión mixta mejora las velocidades de entrenamiento con una pérdida de precisión mínima, especialmente en hardware equipado con Tensor Cores.

12.2 Entrenamiento de múltiples GPU

Nvidia NeMo permite el entrenamiento con múltiples GPU, lo que reduce la duración del entrenamiento para modelos grandes. Esta sección aclara las ventajas de precisión mixta y las distinciones entre entrenamiento con múltiples GPU y múltiples nodos.

12.3 Nvidia NeMo, PyTorch Lightning and Hydra

Al integrar PyTorch Lightning para la eficiencia del entrenamiento e Hydra para la agestión de la configuración, Nvidia NeMo agiliza la investigación de IA conversacional al organizar el código PyTorch y automatizar el flujo de trabajo de entrenamiento.

12.4 Modelos optimizados previamente entrenados

A través de Nvidia GPU Cloud (NGC), Nvidia NeMo ofrece una colección de modelos optimizados y previamente entrenados para diversas aplicaciones de IA conversacional, lo que facilita la integración en proyectos de investigación y proporciona una ventaja en el desarrollo de IA conversacional.

13. Versiones de componentes de software

13.1 Nvidia NeMo Framework 24.05

Componente del Software	PyTorch	PyTorch Lightning	Megatron Core	Transformer Engine	NeMo	NeMo Aligner	NeMo Data Curator
Versión	2.3.0a0	2.2.4	0.7.0	1.6.0.dev0	2.0.0rc0	0.3.0.dev0	0.3.0

13.2 Nvidia NeMo Framework 24.01

Componente del Software	Transformer Engine	PyTorch	NeMo	NeMo Aligner	Megatron Core	PyTorch Lightning	Hydra

Versión	v1.3	2.2.0	1.23.0	0.2.0	0.5.0	2.0.7	1.2.0

Componente del Software	NeMo Data Curator	Kubernetes	Helm	GPU Operator	Network Operator	KubeFlow Operator
Versión	0.1.0	1.27.4	3.12.1	23.3.2	23.1.0	1.6.0

13.3 Nvidia NeMo Framework 23.11

Componente del Software	Transformer Engine	PyTorch	NeMo	Kubernetes	Megatron Core	PyTorch Lightning	Hydra
Versión	v1.1	2.1.0	1.22.0	1.27.4	0.4.0	2.0.7	1.2.0

Componente del Software	Helm	GPU Operator	Network Operator	KubeFlow Operator
Versión	3.12.1	23.3.2	23.1.0	1.6.0

14. Empezando con Nvidia NeMo

14.1 Obtener acceso a Nvidia NeMo Framework

Nvidia NeMo Framework está disponible como contenedor acoplable. Para acceder a él, el usuario debe iniciar sesión o crear una cuenta en Nvidia NGC.

14.2 Modelos multimodales y de lenguaje grande (Large Language and Multimodal models)

Para obtener la versión más reciente de la imagen de Docker (paquete que contiene todo lo necesario para ejecutar una aplicación: el código, las bibliotecas, las dependencias y la configuración necesaria) de NeMo desde el registro de contenedores de Nvidia (ncvr.io), el usuario debe de utilizar la última etiqueta con el formato aa.mm.(parche), por ejemplo:

docker pull nvcr.io/nvidia/nemo:24.05

14.3 IA de voz (Speech AI)

Para obtener la imagen del Docker más reciente de Nvidia NeMo específica para los modelos de inteligencia artificial relacionados con el reconocimiento y síntesis de voz, el usuario debe utilizar la última etiqueta más reciente con el formato aa.mm.speech, por ejemplo:

docker pull nvcr.io/nvidia/nemo:24.05

15. Modelos de lenguaje grandes

15.1 Entrenamiento

Nvidia NeMo puede entrenar LLM de cualquier escala, desde modelos pequeños con unos pocos miles de millones de parámetros hasta modelos muy grandes con cientos de miles de millones y billones de parámetros. Nvidia NeMo aborda todos estos desafío aprovechando cargadores de datos optimizados y escalables, técnicas de paralelismo de modelos, optimizaciones de memoria y proporcionando recetas de capacitación.

Hay dos maneras de entrenar Nvidia NeMo, con o sin Nvidia NeMo Launcher.

- Sin Nvidia NeMo Launcher: Esto funciona bien con una configuración simple que involucra modelos pequeños en uno o varios nodos. Puede ayudar a entrenar más rápidamente y exponer directamente a los guiones de entrenamiento de Nvidia NeMo. Para obtener un tutorial de capacitación concreto paso a paso, Nvidia recomienda seguir la guía de capacitación del modelo GPT.
- Nvidia NeMo Launcher facilita el desarrollo de trabajos de capacitación de modelos en grandes clústeres (como BCM, BCP, AWS, Azure y OCI). Con NeMo Launcher, puedes gestionar y organizar eficientemente tus experimentos en varios nodos. Para usarlo, simplemente editas un archivo de configuración YAML con los detalles de tu trabajo y se ejecuta con un script. Esto simplifica el proceso y oculta las herramientas y scripts complejos subyacentes de Nvidia NeMo, permitiéndote manejar fácilmente la capacitación a gran escala. Para empezar, Nvidia recomienda seguir el manual de entrenamiento previo del modelo Foundation usando Nvidia NeMo Framework.

15.2 Alineación

Nvidia NeMo Aligner es un conjunto de herramientas escalable para una alineación eficiente de modelos. El kit de herramientas admite algoritmos de alineación de modelos de última generación, como SteerLM, DPO y Reinforcement Learning from Human Feedback (RLHF). Estos algoritmos permiten a los usuarios alinear modelos de lenguaje para que sean más seguros, inofensivos y útiles.

El kit de herramientas Nvidia NeMo Aligner está construido utilizando Nvidia NeMo Framework, que permite escalar el entrenamiento hasta miles de GPU utilizando tensor, datos y paralelismo de canalización para todos los componentes de alineación. Todos los puntos de control son compatibles con el ecosistema de Nvidia NeMo, que permite una mayor personalización e implementación de inferencias.

La forma recomendada de utilizar Nvidia NeMo Aligner es a través del contenedor acoplable Nvidia NeMo Framework, como se menciona en obtener acceso a Nvidia NeMo Framework.

Para los desarrolladores de Nvidia NeMo Framework, también es posible instalar Nvidia NeMo Aligner desde el código fuente o crear un contenedor acoplable siguiendo las instrucciones en el repositorio de Nvidia NeMo Aligner Github.

Para obtener un flujo de trabajo paso a paso de RLHF (Reinforcement Learning from Human Feedback) de un extremo a otro en un modelo GPT-2B pequeño, el usuario debe seguir los tutoriales en la documentación de Model Alignment. Las tres fases del RLHF son:

- SFT Training: Ajuste fino supervisado utilizando ejemplos etiquetados para mejorar la capacidad inicial del modelo.
- Reward Model Training: Entrenamiento de un modelo para evaluar automáticamente la calidad de las respuestas basándose en retroalimentación humana.
- PPO Training: Optimización del modelo utilizando aprendizaje por refuerzo para maximizar las recompensas y mejorar la calidad de las respuestas generadas de manera estable y continua.

Además, Nvidia NeMo tiene dos nuevos métodos de alineación:

- SteerLM: técnica basada en SFT (ajuste fino supervisado) condicionada, con salida orientable.
- DPO: algoritmo de alineación liviano en comparación con RLHF (aprendizaje por refuerzo a partir de retroalimentación humana) con una función de pérdida más simple.

15.3 Personalización

La personalización del modelo permite a los usuarios adaptar un LLM general previamente capacitado a un caso de uso o dominio específico. Esto permite producir un modelo ajustado que puede aprovechar la gran cantidad de datos disponibles durante el entrenamiento previo y al mismo tiempo generar resultados que sean más precisos para una tarea posterior específica.

La personalización del modelo se logra ajustando el LLM de forma supervisada. Hay dos métodos:

- Ajuste fino de parámetros completos, conocido en Nvidia NeMo como ajuste fino supervisado (SFT).
- Ajuste fino eficiente en parámetros (PEFT)

En SFT, todos los parámetros del modelo se actualizan para producir resultados que se adaptan a una tarea en concreto. Por otro lado, PEFT ajusta una cantidad mucho menor de parámetros que se insertan en el modelo base en ubicaciones específicas. Si bien SFT suele producir el mejor resultado posible, los métodos PEFT generalmente pueden alcanzar casi el mismo grado de precisión con un costo computacional menor. Dado que los modelos de lenguaje son cada vez más grandes, el método PEFT está ganando popularidad debido a la ligereza en el hardware de capacitación.

El framework de Nvidia NeMo admite SFT y varias técnicas de PEFT. Se puede encontrar más información en la documentación para desarrolladores de Nvidia NeMo.

15.4 Inferencia

Nvidia NeMo Framework proporciona tres maneras distintas para la inferencia de LLM, que se adaptan a diferentes escenarios de implementación y necesidades de rendimiento únicas. Esto incluye inferencias dentro del framework, exportación a TensorRT-LLM e implementación con Triton, e implementación empresarial con Nvidia NIM.

La inferencia dentro del framework implica ejecutar modelos LLM directamente dentro del framework de Nvida NeMo. Este enfoque es sencillo y no requiere exportar modelos a otro formato. Es perfecto para las fases de desarrollo y prueba, donde la facilidad de uso y la flexibilidad son importantes. Nvidia NeMo framework permite la inferencia de múltiples nodos y múltiples GPU, maximizando el

rendimiento. Este método permite iteraciones rápidas y pruebas directamente dentro del entorno de Nvidia NeMo.

Para escenarios que requieren un rendimiento optimizado, los modelos de Nvidia NeMo pueden aprovechar la tecnología de TensorRT-LLM, una biblioteca especializada para acelerar y optimizar la inferencia LLM en las GPUs de Nvidia. Este proceso implica convertir los modelos de Nvida NeMo a un formato compatible con TensorRT-LLM utilizando el módulo nemo.export.

Una vez exportados, estos modelos se pueden implementar utilizando Nvidia Triton Inference Server, una plataforma para implementar modelos de IA entrenados para que sean accesibles a través de una red. Nvidia Triton Inference Server proporciona una forma escalable y eficiente de atender solicitudes de inferencia, admitiendo diversos protocolos como HTTP/REST y gRPC. Puede manejar configuraciones de una sola GPU, múltiples GPUs y múltiples nodos, lo que garantiza un alto rendimiento y una baja latencia para la inferenia de LLM. Nvidia NeMo framework también incluye el módulo nemo.deploy para simplificar este proceso a unas pocas líneas de código.

Las empresas que buscan una solución integral que incluya implementación local o en la nube pueden utilizar Nvidia NIM. Este enfoque aprovecha la suite de Nvidia AI Enterprise, que incluye soporte para Nvidia NeMo, Nvidia Triton Inference Server, TensorRT-LLM y otro software de IA de Nvidia. Esta opción es ideal para organizaciones que necesitan una solución confiable y escalable para implementar modelos de IA generativa en entornos de producción.

16. Modelos Multimodales

Nvidia NeMo framework introduce soporte para modelos multimodales al proporcionar software optimizado para entrenar e implementar modelos SOTA (son sistemas de inteligencia artificial que alcanzan el más alto nivel de rendimiento en tareas que requieren la integración y procesamiento de múltiples tipos de datos o modalidades) en varias categorías: modelos de lenguaje multimodal, fundamentos de visión-lenguaje, modelos de texto a imagen y generación 2D usando NeRF (técnica avanzada de generación de imágenes tridimensionales a partir de una colección de imágenes 2D).

16.1 IA de voz

Nvidia NeMo framework admite el entrenamiento y la personalización de modelos de inteligencia artificial del habla para tareas como el reconocimiento automático de voz (ASR) y la síntesis de texto a voz (TTS).

Para una configuración rápida del entrenamiento y la inferencia de la IA del habla de Nvidia NeMo, es recomendable utilizar el contenedor de voz de Nvidia NeMo framework.

El usuario puede probar la inferencia rápida de los modelos ASR y TTS de Nvidia NeMo con la guía de inicio rápido de Speech AI.

Para obtener más información sobre el entrenamiento del modelo de IA de voz (Speech AI), el usuario puede consultar los cuadernos y los tutoriales.

16.2 Compendio

Los manuales de Nvidia NeMo framework demuestran cómo utilizar el contenedor de capacitación de Nvidia NeMo framework para ajustar los modelos de lenguajes grandes (LLM) con diferentes conjuntos de datos. La información incluye:

- Cómo configurar su infraestructura para utilizar los manuales con DGX Cloud y Kubernetes.
- Utilizar los LLM Llama 2, Mixtral-8x7B y Mistral-7B para preprocesar, entrenar, validar, probar y ejecutar scripts de ajuste.
- Aplicar técnicas de ajuste fino supervisado (SFT) y ajuste fino paramétrico eficiente (PEFT) a los conjuntos de datos databricks-dolly-15k y PubMedQA.
- Configurar y lanzar la capacitación previa del modelo básico en su infraestructura.

16.3 Configuración de infraestructura

El manual Run NeMo Framework on DGX Cloud se centra en preparar un conjunto de datos y entrenar previamente un modelo fundamental con Nvidia NeMo Framework en DGX Cloud. El manual cubre aspectos esenciales de DGX Cloud, como cargar contenedores, crear espacios de trabajo, montar espacios de trabajo, iniciar trabajos y entrenar previamente un modelo.

El manual Run NeMo Framework on Kubernetes (plataforma de código abierto para la automatización del despliegue, el escalado y la gestión de aplicaciones en contenedores) demuestra la implementación y administración de Nvidia NeMo usando Kubernetes. El manual cubre la configuración del clúster, la instalación de Nvidia NeMo Framework, la preparación de datos y el entrenamiento de modelos.

16.4 Alineación del Modelo

- El manual de Nvidia NeMo Framework SFT con Llama 2 muestra cómo ajustar modelos de Llama 2 de varios tamaños usando SFT en comparación

con el conjunto de datos databricks-dolly-15k. Demuestra el preprocesamiento de datos, el entrenamiento, la validación, las pruebas y la ejecución de los scripts de ajuste incluidos en Nvidia NeMo Framework. También muestra cómo realizar inferencias contra el modelo ajustado.

- El manual de Nvidia NeMo Framework con Mistral-7B muestra cómo ajusta el modelo Mistral-7B usando SFT con el conjunto de datos databricks-dolly-15k. Demuestra el preprocesamiento de datos, el entrenamiento, la validación, las pruebas y la ejecución de los scripts de ajuste incluidos en Nvidia NeMo Framework.
- El manual de Nvidia NeMo Framework SFT con Mixtral-8x7B muestra cómo ajustar Mixtral 8x7B usando SFT con el conjunto de datos databricks-dolly-15k. Demuestra el preprocesamiento de datos, el entrenamiento, la validación, las pruebas y la ejecución de los scripts de ajuste incluidos en Nvidia NeMo Framework. También muestra cómo realizar inferencias contra el modelo ajustado.
- El manual de Nvidia NeMo Framework PEFT con Mistral-7B muestra cómo ajustar el modelo Mistral-7B usando PEFT con el conjunto de datos PubMedQA. Demuestra el preprocesamiento de datos, el entrenamiento, la validación, las pruebas y la ejecución de los scripts de ajuste incluidos en Nvidia NeMo Framework. También muestra cómo realizar inferencias contra el modelo ajustado.
- El manual de Nvidia NeMo Framework PEFT muestra cómo ajustar los modelos Mixtral 8x7B y Llama 2 de varios tamaños usando PEFT en comparación con el conjunto de datos PubMedQA. Demuestra el preprocesamiento de datos, el entrenamiento, la validación, las pruebas y la ejecución de los scripts de ajuste incluidos en Nvidia NeMo Framework.

16.5 Preentrenamiento

- El manual de preentrenamiento del modelo básico de Nvidia NeMo Framework se centra en lanzar eficazmente un trabajo de preentrenamiento del modelo básico en su infraestructura y obtener las herramientas de entrenamiento necesarias como resultado de ejecuciones exitosas. Demuestra cómo ejecutar el flujo de trabajo de modelos básicos previos al entrenamiento utilizando Nvidia NeMo Framework y el conjunto de datos Pile (colección de datos a gran escala, específicamente diseñada para entrenar y evaluar modelos de procesamiento de lenguaje natural (NLP)), además de producir puntos de control, registros y archivos de eventos.
- El manual de Nvidia NeMo Framework AutoConfigurator demuestra cómo utilizar Nvidia NeMo Framework AutoConfiguratos para determinar el tamaño de modelo óptimo para un presupuesto de computación y capacitación determinado. También muestra cómo producir configuraciones de inferencia y preentrenamiento del modelo básico óptimo para lograr ejecuciones de mayor

rendimiento. Se centra específicamente en automatizar el proceso de configuración de NeMo, como la configuración automática, el ajuste de parámetros y la optimización para agilizar la configuración.
- El manual de preentrenamiento de un solo nodo de Nvidia NeMo Framework muestra cómo entrenar previamente un modelo simple de estilo GPT utilizando hardware de específico.

17. SFT y PEFT

La personalización de modelos permite al usuario adaptar un LLM general previamente capacitado a un caso de uso o dominio específico. Este proceso da como resultado un modelo ajustado que se beneficia de los extensos datos de preentrenamiento, al mismo tiempo que genera resultados más precisos. La personalización del modelo se logra mediante ajustes supervisados y se divide en dos categorías populares:

- Ajuste fino de parámetros completos, que en Nvidia NeMo se conoce como ajuste fino supervisado (SFT)
- Ajuste fino eficiente en parámetros (PEFT)

En SFT, todos los parámetros del modelo se actualizan para producir resultados que se adaptan a la tarea.

PEFT, por otro lado, ajusta una cantidad mucho menor de parámetros que se insertan en el modelo base en ubicaciones estratégicas. Al realizar un ajuste fino con PEFT, los pesos del modelo base permanecen congelados y solo se entrenan los módulos adaptadores (en vez de entrenar todo el modelo, PEFT añade pequeños módulos llamados "adaptadores" en puntos estratégicos del modelo base. Solo estos adaptadores se entrenan y ajustan). Como resultado, el número de parámetros entrenables se reduce significativamente (< 1%).

Si bien SFT suele producir los mejores resultados posibles, los métodos PEFT pueden lograr casi el mismo grado de precisión y, al mismo tiempo, reducir significativamente el costo computacional. A medida que los modelos de lenguaje continúan creciendo en tamaño, PEFT está ganando popularidad debido a sus requisitos menores en el hardware de capacitación.

Nvidia NeMo admite SFT y cuatro métodos PEFT que se pueden utilizar con varios modelos basados en transformadores. En el cuadro de abajo hay una colección de scripts de conversión que convierten modelos populares del formato HF (Hugging Face, compañía conocida por su trabajo en procesamiento de lenguaje natural (NLP) y aprendizaje automático) al formato Nvidia Nemo.

	GPT 3	Nemotron	LLaMa 1/2	Falcon	Starcoder	Mistral	Mixtral	Gemma	T5
SFT	✅	✅	✅	✅	✅	✅	✅	✅	✅
LoRA	✅	✅	✅	✅	✅	✅	✅	✅	✅
P-tuning	✅	✅	✅	✅	✅	✅	✅	✅	✅
Adapters (Canonical)	✅	✅	✅		✅	✅	✅	✅	✅
IA3	✅	✅	✅		✅	✅		✅	✅

17.1 Terminología: PEFT frente a Adaptador

El término "PEFT" se usa para describir el método de ajuste fino eficiente en parámetros. Además, se utiliza el término "adaptador" para referirse al módulo suplementario inyectado en un modelo base congelado (significa mantener sus parámetros (pesos y sesgos) sin cambios durante un nuevo proceso de entrenamiento). Cada modelo PEFT tiene la flexibilidad de utilizar uno o más tipos de adaptadores.

Entre los diferentes métodos PEFT, un método a veces se denomina "adaptadores" porque fue uno de los primeros usos propuestos de módulos adaptadores en PNL (Procesamiento de Lenguaje Natural). Para diferenciar entre los dos usos, nos referiremos a este método PEFT como adaptadores "canónicos".

17.2 Cómo PEFT funciona en los modelos de Nvidia NeMo

Cada método PEFT tiene uno o más tipos de adaptadores que deben inyectarse en el modelo base. En los modelos NeMo, la lógica del adaptador y los pesos (parámetros ajustables de los módulos adaptadores que se añaden a un modelo base congelado) del adaptador ya están integrados en los submódulos, pero están deshabilitados de forma predeterminada para el entrenamiento y configuración.

Al realizar PEFT, la ruta lógica del adaptador se puede habilitar cuando se llama a model.add_adapter(peft_cfg). En esta función, el modelo escanea cada adaptador aplicable en busca del método PEFT actual, examinando sus submódulos para identificar las rutas lógicas del adaptador que se pueden habilitar.

Posteriormente, los pesos del modelo base se congelan, mientras que los pesos del adaptador recién agregados permanecen descongelados y se pueden actualizar

durante el ajuste, lo que genera ganancias de eficiencia en la cantidad de parámetros ajustados.

17.3 Clases de configuración PEFT

Cada método PEFT está especificado por una clase PEFTConfig que almacena los tipos de adaptadores aplicables al método PEFT, así como los hiperparámetros necesarios para inicializar estos módulos adaptadores.

See admiten los siguientes cuatro métodos PEFT:

1. Adapters (canonical): CanonicalAdaptersPEFTConfig
2. LoRA: LoraPEFTConfig
3. IA3: IA3PEFTConfig
4. P-Tuning: PtuningPEFTConfig

Estas clases de configuración simplifican la experimentación con diferentes adaptadores al permitir cambios sencillos en la clase de configuración.

17.4 Clases de modelo base

PEFT en NeMo está construido con una clase mixta que no pertenece a ningún modelo en particular. Esto significa que la misma interfaz está disponible para diferentes modelos de Nvidia NeMo. Actualmente, Nvidia NeMo admite PEFT para modelos estilo GPT como GPT 3, Nemotron, LLaMa 1/2 (MegatronGPTSFTModel), así como T5 (MegatronT5SFTModel).

17.5 Ajuste completo frente a PEFT

Puede cambiar entre ajuste completo y PEFT eliminando llamadas a add_adapter y load_adapter. El siguiente fragmento de código ilustra la API principal de ajuste completo y PEFT:

```
trainer = MegatronTrainerBuilder(config).create_trainer()
model_cfg =
MegatronGPTSFTModel.merge_cfg_with(config.model.restore_from_path,
config)
```

```
### Training API ###
model = MegatronGPTSFTModel.restore_from(restore_path, model_cfg,
trainer) # restore from pretrained ckpt
+ peft_cfg = LoraPEFTConfig(model_cfg)
+ model.add_adapter(peft_cfg)
trainer.fit(model)  # saves adapter weights only

### Inference API ###
# Restore from base then load adapter API
model = MegatronGPTSFTModel.restore_from(restore_path, trainer,
model_cfg)
+ model.load_adapters(adapter_save_path, peft_cfg)
model.freeze()
trainer.predict(model)
```

17.6 Métodos PEFT admitidos

1. Adaptadores (canónicos): aprendizaje por transferencia eficiente en parámetros para PNL.
    - Adaptadores (configuración de Houlsby (método específico de incorporar adaptadores en modelos de lenguaje preentrenados)) es uno de los primeros métodos PEFT aplicados a la PNL. El ajuste del adaptador es más eficiente que el ajuste fino completo porque los pesos del modelo base están congelados, mientras que solo se actualiza una pequeña cantidad de pesos del módulo adaptador. En este método se insertan dos capas lineales con un cuello de botella (punto de restricción que limita el rendimiento o la eficiencia de un sistema) y una activación no lineal en cada capa del transformador a través de una conexión residual. En cada caso, la capa lineal de salida se inicializa en 0 para garantizar que un adaptador no capacitado no afecte el paso directo normal de la capa del transformador.
    - En Nvidia NeMo, puede personalizar la dimensión del cuello de botella del adaptador, la cantidad de caída del adaptador, así como el tipo y la posición de la capa de normalización.

2. LoRA: LoRA: Adaptación de bajo rango de modelos de lenguaje grandes.

    - LoRA (Low-Rank Adaptation) hace que el ajuste fino de modelos de inteligencia artificial sea más eficiente usando dos matrices pequeñas en lugar de actualizar todos los pesos del modelo. Los pesos originales del modelo se mantienen sin cambios (congelados), mientras que solo se ajustan estas matrices pequeñas para adaptarse a los nuevos datos, reduciendo así la cantidad de parámetros que se necesitan entrenar. A diferencia de otros métodos como los adaptadores, en LoRA los pesos originales y los nuevos se combinan durante la inferencia sin añadir complejidad o latencia al modelo. Esto permite adaptar el modelo a nuevas tareas de manera eficiente y sin cambiar su estructura básica.
    - En Nvidia NeMo, puede personalizar la dimensión del cuello de botella del adaptador y los módulos de destino para aplicar LoRA. LoRA se puede aplicar a cualquier capa lineal. Para QKV (son matrices usadas en el mecanismo de atención de los transformadores (tipo de modelo de aprendizaje automático que ha revolucionado el procesamiento del lenguaje natural)), la implementación de atención de Nvidia NeMo fusiona QKV en una sola proyección, por lo que nuestra implementación LoRA aprende una única proyección de bajo rango para QKV combinada.

3. IA3: El ajuste fino eficiente de parámetros (PEFT) es mejor y más económico que el aprendizaje en contexto porque permite entrenar el modelo más rápidamente y con menos recursos.

    - IA3 facilita el ajuste fino de modelos de IA usando pequeños vectores que ajustan las activaciones del modelo. Estos vectores se añaden en ciertas partes del modelo y solo ellos se actualizan durante el ajuste fino, lo que reduce la cantidad de parámetros a entrenar. A diferencia de otros métodos que usan matrices grandes, IA3 usa estos pequeños vectores, disminuyendo aún más los parámetros necesarios. Además, estos vectores se integran sin cambiar la estructura del modelo ni añadir tiempo extra durante la inferencia. No necesitas ajustar

manualmente ningún hiperparámetro adicional, simplificando aún más el proceso.

4. P-Tuning: GPT también lo entiende.

    - P-tuning es un método de aprendizaje rápido en el que se añaden elementos especiales, llamados tokens virtuales, al mensaje de entrada del modelo para ayudarlo a realizar una tarea específica. Estos tokens virtuales no son palabras reales ni tienen un significado concreto, sino que son vectores numéricos que el modelo puede entrenar. Aunque no representan palabras reales, tienen la misma forma que los tokens (palabras o partes de palabras) que el modelo ya conoce. Esta técnica permite ajustar el comportamiento del modelo de manera eficiente sin cambiar su estructura básica.
    - En p-tuning, se usa un modelo MLP (perceptrón multicapa) intermedio para crear los tokens virtuales. Este modelo intermedio se llama Prompt_encoder (codificador de aviso). Al principio del entrenamiento, los parámetros del Prompt_encoder se configuran con valores aleatorios. Luego, durante el entrenamiento, los parámetros del modelo principal se mantienen sin cambios (congelados), y solo se actualizan los pesos del Prompt_encoder en cada paso. Esto permite que el Prompt_encoder aprenda a generar tokens virtuales que ayudan al modelo principal a realizar mejor la tarea específica.
    - En Nemo, puede personalizar la cantidad de tokens virtuales, así como las dimensiones de incrustación y cuello de botella de MLP.

18. Modelos de lenguaje grandes

Nvidia NeMo Framework tiene todo lo necesario para entrenar modelos de lenguajes grandes. Esto incluye configurar el clúster (conjunto de computadoras interconectadas que trabajan juntas como si fueran una sola máquina) de computación, descargar datos y seleccionar hiperparámetros del modelo. Las configuraciones predeterminadas para cada modelo y tarea se prueban periódicamente y cada configuración se puede modificar para entrenar en nuevos conjuntos de datos o probar nuevos hiperparámetros del modelo.

19. Modelos Multimodales

NeMo Framework ofrece un sólido soporte para modelos multimodales, ampliando sus capacidades en cuatro categorías clave: modelos de lenguaje multimodal, modelos básicos de visión-lenguaje, modelos de texto a imagen y campos de radiación neuronal (NeRF).

19.1 Modelos de lenguaje multimodal

Los modelos de lenguaje multimodal se centran en enriquecer los modelos de lenguaje con capacidades multimodales, principalmente a través de codificadores visuales (es un componente de un sistema de inteligencia artificial que convierte imágenes o datos visuales en una representación numérica que puede ser procesada por un modelo de aprendizaje automático), para crear modelos que permitan la comprensión visual y textual interactiva. Los modelos compatibles incluyen:

- NeVa (LLaVA): Proporciona capacidades de capacitación, ajuste e inferencia.
- VideoNeVA (LLaVA): Proporciona capacidades de capacitación e inferencia para la modalidad de video.

19.2 Modelos básicos de visión y lenguaje

Con el objetivo de crear modelos capaces de procesar y comprender información tanto visual como textual, los modelos Vision-Language Foundation se pueden ajustar para realizar tareas relacionadas con la visión, como la clasificación y la agrupación. Además, pueden funcionar en modelos de lenguaje visual, modelos de texto a imagen y más. El modelo soportado es:

- CLIP: ofrece capacidades de capacitación e inferencia, sobresaliendo en clasificación de imágenes de disparo cero y puntuación de similitud.
- Modelo de filtrado de contenido NSFW (ajustado en CLIP): proporciona una solución de filtrado basada en visión para identificar contenido explícito.

19.3 Modelos de texto a imagen

Los modelos de texto a imagen están diseñados para generar imágenes a partir de descripciones textuales.

- Modelos de Cimentación: Difusión Estable, Imagen
- Ajuste de modelos: DreamBooth, ControlNet, instructPix2Pix

19.4 Más allá de la generación 2D usando NeRF

NeMo NeRF se concentra en la creación y manipulación de modelos 3D y 4D a través de un enfoque modular, admitiendo modelos innovadores como:

- DreamFusion: este modelo genera objetos 3D detallados a partir de descripciones de texto, utilizando modelos 2D de difusión de texto a imagen previamente entrenados y campos de radiación neuronal (técnica avanzada para representar y renderizar objetos en 3D. Utilizan redes neuronales para aprender cómo la luz interactúa con la superficie de los objetos, creando una representación detallada y realista en 3D) para la renderización.

20. Implementación de modelos de Nvidia NeMo Framework

Nvidia NeMo Framework ofrece varias rutas de implementación para los modelos Nvidia NeMo, adaptadas a diferentes dominios, como modelos de lenguaje grande (LLM) y modelos multimodales (MM). Hay tres rutas de implementación principales para los modelos NeMo: implementación a nivel empresarial con NVIDIA Inference Microservice (NIM), inferencia optimizada mediante la exportación a otra biblioteca e implementación con Triton, e inferencia en el framework.

Dominio	Nvidia NIM	Optimizado	En el framework
LLMs	Sí	N/A	N/A
MMs	N/A	N/A	N/A

21. Rutas disponibles para trabajar con LLM

21.1 Nvidia NIM para LLMs

Las empresas que buscan una solución integral que cubra la implementación local y en la nube pueden utilizar Nvidia NIM. Este enfoque aprovecha Nvidia AI Enterprise, que incluye soporte para NVIDIA NeMo, Triton Inference Server, TensorRT-LLM y otro software de IA de NVIDIA.

Esta opción es ideal para organizaciones que quieren implementar modelos de IA generativa en entornos de producción. También se destaca como la opción de inferencia más rápida, ya que ofrece scripts y API fáciles de usar. Aprovechando el backend (parte de un sistema informático que maneja la lógica, el almacenamiento de datos y la comunicación entre los diferentes componentes del sistema) de TensorRT-LLM Triton, logra una inferencia rápida utilizando algoritmos de procesamiento por lotes avanzados, incluido el procesamiento por lotes en vuelo. Tenga en cuenta que esta ruta de implementación solo admite modelos LLM seleccionados.

21.2 Inferencia dentro del framework

La inferencia dentro del framework implica ejecutar modelos LLM directamente dentro del framework de Nvidia NeMo. Este enfoque es sencillo y elimina la necesidad de exportar modelos a otro formato. Es ideal para las fases de desarrollo y prueba, donde la facilidad de uso y la flexibilidad son fundamentales. Nvidia NeMo Framework admite la inferencia de múltiples nodos y múltiples GPU, mejorando el rendimiento. Este método permite iteraciones rápidas y pruebas directas dentro del entorno de Nvidia NeMo. Aunque esta es la opción más lenta, brinda soporte para todos los modelos Nvidia NeMo.

21.3 Inferencia optimizada para LLM usando TensorRT-LLM

Para escenarios que requieren un rendimiento optimizado, los modelos de Nvidia NeMo pueden aprovechar TensorRT-LLM, una biblioteca especializada para

acelerar y optimizar la inferencia LLM en las GPU NVIDIA. Este proceso implica convertir los modelos Nvidia NeMo a un formato compatible con TensorRT-LLM utilizando el módulo nemo.export. A diferencia de Nvidia NIM para la ruta de LLM, esta opción no incluye los algoritmos de procesamiento por lotes avanzados, como el procesamiento por lotes en vuelo utilizando el backend TensorRT-LLM Triton, que logra la inferencia de LLM más rápida.

22. Guía del usuario

NVIDIA NeMo Framework es un framework nativo de la nube de extremo a extremo para crear, personalizar e implementar modelos de IA generativa en cualquier lugar. Permite la creación de modelos de última generación, incluidos el habla, el lenguaje y la visión.

El entrenamiento de arquitecturas de IA generativa generalmente requiere importantes datos y recursos informáticos. NeMo utiliza PyTorch Lightning para un entrenamiento de precisión mixta multi-GPU/multi-nodo eficiente y de alto rendimiento. NeMo se basa en el potente Megatron-LM y Transformer Engine de Nvidia para sus modelos de lenguajes grandes (LLM) y modelos multimodales (MM), aprovechando avances en entrenamiento y optimización de modelos. Para aplicaciones de voz AI, reconocimiento automático de voz (ASR) y texto a voz (TTS), Nvidia NeMo está desarrollado con PyTorch y PyTorch Lightning, lo que garantiza una integración perfecta y facilidad de uso.

NVIDIA NeMo Framework presenta colecciones separadas para modelos de lenguajes grandes (LLM), modelos multimodales (MM), visión por computadora (CV), reconocimiento automático de voz (ASR) y modelos de texto a voz (TTS). Cada colección tiene módulos prediseñados que incluyen todo lo necesario para entrenar con sus datos. Estos módulos se pueden personalizar, ampliar y componer fácilmente para crear nuevas arquitecturas de modelos de IA generativa.

22.1 Requisitos

1. Versión de Python 3.10 o superior.
2. Versión de Pytorch 1.13.1 o 2.0+.
3. Acceso a una GPU Nvidia.

22.2 Instalación

-Utilizando el contenedor Nvidia PyTorch:

Es recomendable utilizar el contenedor Nvidia PyTorch para aprovechar todas las optimizaciones para la capacitación de LLM, incluido 3D Model Parallel, kernels fusionados (técnica de optimización en el contexto de la computación paralela y el procesamiento de datos, especialmente en el uso de unidades de procesamiento gráfico (GPU)), FP8, entre otros. Para instalarlo:

```
docker pull nvcr.io/nvidia/pytorch:24.01-py3
docker run --gpus all -it nvcr.io/nvidia/pytorch:24.01-py3
```

Sin el contenedor, puedes instalar Nvidia NeMo y sus dependencias:

```
apt-get update && apt-get install -y libsndfile1 ffmpeg
pip install Cython
pip install nemo_toolkit['all']
```

-Instalación del motor transformador:

Este paso implica clonar el repositorio de Transformer Engine, verificar una confirmación específica e instalarla con indicadores específicos.

```
git clone https://github.com/NVIDIA/TransformerEngine.git && \
cd TransformerEngine && \
git fetch origin 8c9abbb80dba196f086b8b602a7cf1bce0040a6a && \
git checkout FETCH_HEAD && \
git submodule init && git submodule update && \
NVTE_FRAMEWORK=pytorch NVTE_WITH_USERBUFFERS=1 MPI_HOME=/usr/local/mpi pip install .
```

-Instalación Apex

```
git clone https://github.com/NVIDIA/apex.git && \
cd apex && \
git checkout c07a4cf67102b9cd3f97d1ba36690f985bae4227 && \
cp -R apex /usr/local/lib/python3.10/dist-packages
```

-Instalación PyTorch Lightning

Este paso implica instalar una versión corregida de errores de PyTorch Lightning desde una rama específica.

```
git clone -b bug_fix https://github.com/athitten/pytorch-lightning.git && \
cd pytorch-lightning && \
PACKAGE_NAME=pytorch pip install -e .
```

-Instalación Megatron Core

Esta código ayuda a detallar los pasos para clonar e instalar Megatron Core.

```
git clone https://github.com/NVIDIA/Megatron-LM.git && \
cd Megatron-LM && \
git checkout a5415fcfacef2a37416259bd38b7c4b673583675 && \
pip install .
```

-Instalación Model Optimizer

Este último paso implica instalar el paquete Model Optimizer.

```
pip install nvidia-modelopt[torch]~=0.11.0 --extra-index-url https://pypi.nvidia.com
```

```
apt-get update && apt-get install -y libsndfile1 ffmpeg
pip install Cython
pip install nemo_toolkit['all']
```

-Instalación de Conda

Si no se utiliza el contenedor Nvidia PyTorch, Nvidia recomienda instalar Nvidia NeMo en un entorno Conda (es un sistema de gestión de paquetes y un entorno de gestión de software de código abierto que se utiliza principalmente para la instalación y gestión de bibliotecas y dependencias en los lenguajes de programación Python y R).

```
conda create --name nemo python==3.10.12
conda activate nemo
```

22.3 Instalación en Google Colab

Lo primero que el usuario debe hacer es crear un nuevo documento de Jupyter Notebook en Google Colab. Una vez creado, debe copiar el siguiente código:

```
BRANCH = 'v1.0.0'
!python -m pip install
git+https://github.com/NVIDIA/NeMo.git@$BRANCH#egg=nemo_toolkit[all]
```

Una vez escrito el código, debemos de escribir el siguiente para instalar todos los paquetes necesarios para su funcionamiento:

```
# Import NeMo and it's ASR, NLP and TTS collections
import nemo
# Import Speech Recognition collection
import nemo.collections.asr as nemo_asr
# Import Natural Language Processing colleciton
import nemo.collections.nlp as nemo_nlp
# Import Speech Synthesis collection

import nemo.collections.tts as nemo_tts
# We'll use this to listen to audio
import IPython
```

Bloque Práctico

23. Prácticas con la Tarjeta Gráfica usando Google Colab

- Audio a Texto: Este programa utiliza herramientas de NVIDIA NeMo para convertir audio a texto. Primero, se basa en un modelo de reconocimiento de voz automática (ASR) llamado QuartzNet, que es muy eficiente para transcribir audio en inglés. Este modelo viene preentrenado y se carga usando la biblioteca NeMo de NVIDIA, lo que permite transformar una grabación de audio en texto de forma automática. Si el audio está en línea, el programa puede descargarlo usando la biblioteca requests, luego ajusta la calidad del audio con torchaudio para asegurarse de que esté en la frecuencia correcta (16 kHz), y finalmente el modelo QuartzNet genera la transcripción del audio, devolviendo el texto hablado.
- Texto a Audio: Para la conversión de texto a audio, el programa utiliza el modelo Tacotron2, también proporcionado por NeMo, que convierte el texto escrito en un espectrograma, es decir, una representación visual del sonido. Luego, para generar el sonido real a partir de este espectrograma, se usa un modelo llamado WaveGlow, que convierte el espectrograma en una onda sonora (audio). El resultado es un archivo de audio donde una voz artificial reproduce el texto que se le dio. Todo el proceso se gestiona con las herramientas de NeMo y se guarda el archivo de audio usando torchaudio, una biblioteca que maneja archivos de sonido. De esta forma, el programa permite generar voz artificial a partir de texto.

  .

23.1 Audio a texto y texto a audio

```
!pip install nemo_toolkit['all'] #1
!pip install nemo_text_processing #2
!pip install pytorch-lightning #3
!pip install torchaudio #4
```

#1 !pip install nemo_toolkit['all']:

- NeMo Toolkit es una biblioteca desarrollada por NVIDIA que proporciona herramientas para modelos de reconocimiento de voz (ASR - Automatic Speech Recognition (en español: Reconocimiento de Habla Automático)), síntesis de voz (TTS - Text-to-Speech (en español: Texto a Habla)), y otras tareas relacionadas con el procesamiento de audio y lenguaje natural. El argumento 'all' asegura que todas las funcionalidades y componentes de NeMo se instalen, incluidas las colecciones de ASR y TTS.

#2 !pip install nemo_text_processing:

- Instala el módulo de procesamiento de texto de NeMo. Este módulo se utiliza para tareas como normalización de texto (convertir números a palabras, como “123” a “ciento veintitrés”), tokenización (dividir el texto en partes más pequeñas, como palabras o sílabas), y otras transformaciones necesarias en aplicaciones que usan ASR y TTS. Es especialmente útil en sistemas de síntesis de voz (TTS) donde el texto debe ser procesado antes de ser convertido en audio.

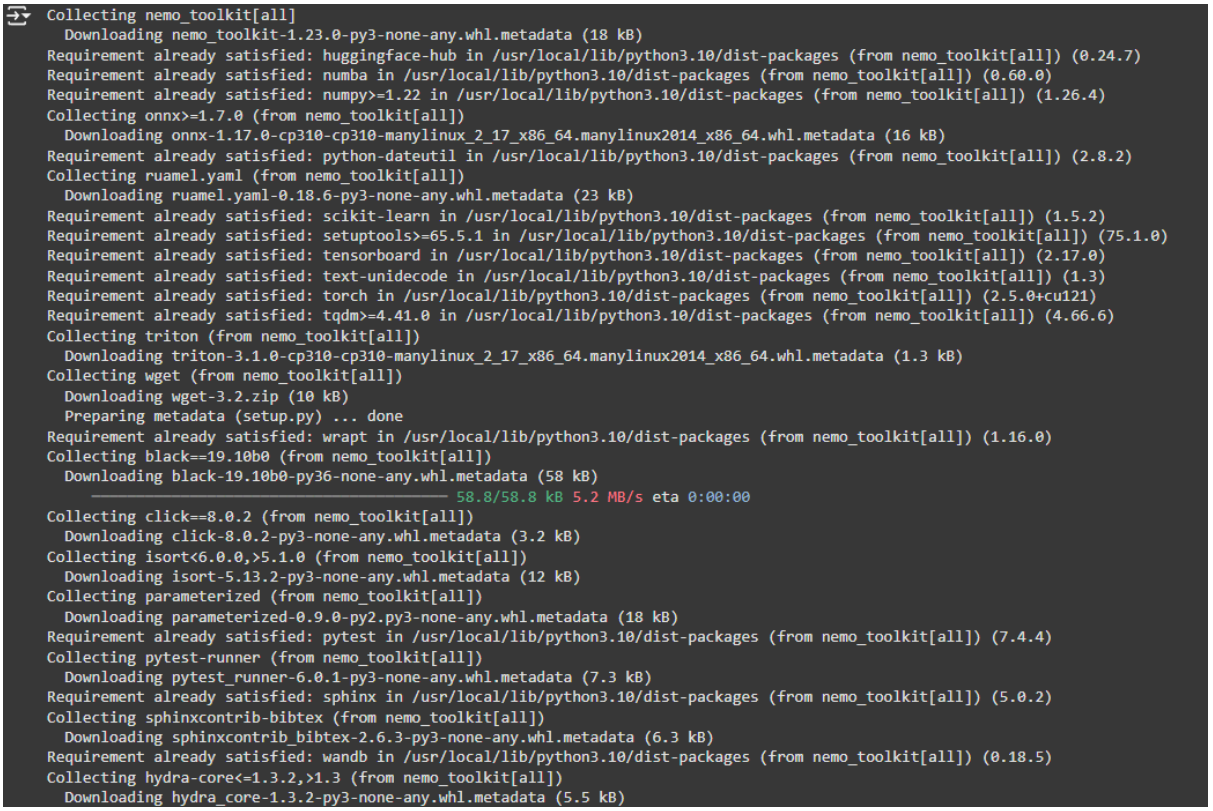
#3 !pip install pytorch-lightning:

- Instala PyTorch Lightning, una biblioteca de alto nivel que organiza y optimiza el código de PyTorch (es una biblioteca de aprendizaje profundo que facilita la creación y entrenamiento de modelos de inteligencia artificial, ofreciendo herramientas flexibles para trabajar con redes neuronales). Facilita la implementación de modelos de deep learning (aprendizaje profundo), ofreciendo una estructura más organizada y funcionalidades avanzadas como el entrenamiento distribuido (el entrenamiento distribuido acelera el entrenamiento de modelos dividiendo la tarea entre múltiples dispositivos. Se puede hacer repartiendo datos o partes del modelo entre varios procesadores), manejo de GPU, y más.

#4 !pip install torchaudio:

- Instala Torchaudio, una biblioteca complementaria de PyTorch para el procesamiento de señales de audio. Permite cargar, transformar, y manipular archivos de audio, lo cual es esencial para tareas de reconocimiento y síntesis de voz.

Una vez instaladas las librerias y dependencias necesarias para mi IA, saldrá una pestaña con la información siguiente:



En esta captura de pantalla se puede ver como se van insalando lo necesario. En esta captura no muestro todo lo que se instala porque ocupa bastante espacio, así que solo dejo este pequeño ejemplo con lo más importante.

```
import sys #5

# Definimos una clase ModelFilter ficticia para satisfacer el
requisito de importación
class ModelFilter: #6
    pass  # Agregamos cualquier atributo o método necesario si lo
requiere NeMo

# Agregamos el ModelFilter ficticio a sys.modules
sys.modules['huggingface_hub.ModelFilter'] = ModelFilter #7
```

#5 import sys:

- El módulo sys proporciona acceso a variables y funciones que interactúan con el entorno del lenguaje de programación. Python. Lo usamos para modificar el diccionario sys.modules.

#6 class ModelFilter::

- Se define una clase vacía ModelFilter que no hace nada más que satisfacer la necesidad de importar esta clase. Esto es un "truco" para evitar errores de importación cuando NeMo intenta cargar el módulo huggingface_hub.ModelFilter, el cual no existe y es imposible de descargar.

#7 sys.modules['huggingface_hub.ModelFilter'] = ModelFilter:

- Aquí se añade manualmente la clase ModelFilter al diccionario sys.modules, que es donde Python guarda los módulos que ya han sido importados. Esto asegura que cuando NeMo intente importar huggingface_hub.ModelFilter, encuentre esta clase ficticia y el código pueda continuar sin problemas, permitiendo que podamos descargar bien los paquetes necesarios para que podamos usar Nvidia NeMo.

```
# Ahora podemos importat Nvidia NeMo
import nemo.collections.asr as nemo_asr #8
import nemo.collections.tts as nemo_tts #9
import torchaudio #10
import requests #11
```

#8 nemo.collections.asr:

- Esta línea importa la colección de modelos de Reconocimiento Automático de Voz (ASR) de NeMo y la asigna al alias nemo_asr. ASR (Automatic Speech Recognition) es la tecnología que permite convertir el audio en texto. NVIDIA NeMo incluye varios modelos preentrenados para ASR, como QuartzNet, que están optimizados para transcribir diferentes idiomas (como inglés y español) y permiten un rendimiento de alta precisión en tareas de transcripción de voz.

#9 nemo.collections.tts:

- Esta línea importa la colección de modelos de Síntesis de Voz (TTS) de NeMo, y se le asigna el alias nemo_tts. TTS (Text-To-Speech) convierte texto en audio de voz, y NeMo incluye modelos como Tacotron2 y WaveGlow. Estos modelos permiten generar audio realista a partir de texto en varios idiomas. En este código, se utiliza Tacotron2 para generar el mel-espectrograma (es una representación visual de las características del

sonido, especialmente diseñada para capturar la percepción humana de las frecuencias) y WaveGlow para convertir ese espectrograma en audio.

#10 torchaudio:

- “torchaudio” es una biblioteca que forma parte del ecosistema PyTorch para manejar y procesar datos de audio. En este código se utiliza para varias tareas. Puede cargar archivos de audio: permite leer archivos de audio en diferentes formatos y extraer datos como el audio en sí y la tasa de muestreo. Segundo, transformaciones: con torchaudio.transforms, es posible aplicar transformaciones como la resampleación para cambiar la frecuencia de muestreo del audio, que es necesario para asegurarse de que el audio esté en la frecuencia correcta que espera el modelo (por ejemplo, 16 kHz para el ASR de NeMo). Y por último, guardar audio: facilita guardar el audio generado (por ejemplo, después de convertir texto a audio) en un archivo de formato WAV o similar.

#11 requests:

- “requests” es una biblioteca para realizar solicitudes HTTP en Python. Aquí lo estamos utilizando para descargar archivos de audio desde una URL en caso de que el archivo de entrada para la transcripción esté en Internet. Si el archivo de audio es una URL, requests descarga el archivo y lo guarda localmente como temp_audio.wav. Luego, este archivo local es procesado y convertido a texto por el modelo de ASR.

Una vez ejecutado el código, se mostrará un mensaje en la terminal comforme se ha ejecutado correctamente.

```
nemo_asr.models.EncDecCTCModel.list_available_models() #12
```

#12 nemo_asr.models.EncDecCTCModel.list_available_models()

El método nemo_asr.models.EncDecCTCModel.list_available_models() se utiliza para ver una lista de todos los modelos preentrenados disponibles dentro de NVIDIA NeMo para la tarea de reconocimiento automático de voz (ASR) que usan la arquitectura de codificador-decodificador (EncDec) con CTC (Connectionist Temporal Classification).

Este método permite ver qué modelos de reconocimiento de voz (ASR) puedo usar sin necesidad de recordar o buscar sus nombres. Además, permite comparar diferentes modelos preentrenados que son adecuados para distintas tareas o idiomas. Dependiendo del caso de uso, podemos elegir el modelo que mejor se ajusta a nuestras necesidades.

Al darle a ejecutar esta parte del código se mostrará en la terminal todos los modelos de reconocimiento de voz (ASR). Así, se puede ver todos los modelos disponibles sin necesidad de buscar en internet o recordar los nombres. Aquí muestro un pequeño fragmento de lo que debería salir en la terminal:

```
pretrained_models = [
    PretrainedModelInfo(
        pretrained_model_name="QuartzNet15x5Base-En",
        description=(
            "QuartzNet15x5 model trained on six datasets:
LibriSpeech, Mozilla Common Voice "
            "(validated clips from en_1488h_2019-12-10), WSJ,
Fisher, Switchboard, and NSC Singapore English. "
            "It was trained with Apex/Amp optimization level O1
for 600 epochs. The model achieves a WER of 3.79% "
            "on LibriSpeech dev-clean, and a WER of 10.05% on
dev-other. Please visit "

"https://ngc.nvidia.com/catalog/models/nvidia:nemospeechmodels for
further details."
        ),

location="https://api.ngc.nvidia.com/v2/models/nvidia/nemospeechmo
dels/versions/1.0.0a5/files/QuartzNet15x5Base-En.nemo"
    ),
    PretrainedModelInfo(
        pretrained_model_name="stt_en_quartznet15x5",
        description="For details about this model, please visit
https://ngc.nvidia.com/catalog/models/nvidia:nemo:stt_en_quartznet
15x5",

location="https://api.ngc.nvidia.com/v2/models/nvidia/nemo/stt_en_
quartznet15x5/versions/1.0.0rc1/files/stt_en_quartznet15x5.nemo"
    ),
```

En los siguientes códigos de abajo, se explica la creación del algoritmo para transcribir audio a texto.

```
"""
stt_en_quartznet15x5: Este es un modelo QuartzNet que ha
demostrado ser muy eficiente para el reconocimiento de voz en
inglés. Es una buena opción si tu tarea principal es transcribir
audio en inglés.

stt_en_jasper10x5dr: Jasper es otra arquitectura de red que se ha
utilizado para ASR. Este modelo es adecuado si necesitas un modelo
alternativo al QuartzNet para la transcripción en inglés.

stt_es_quartznet15x5: Este modelo es específico para español. Si
tu audio está en español, este es el modelo que debes utilizar."""

# Configuración para convertir Audio a Texto (ASR)
asr_model =
nemo_asr.models.EncDecCTCModel.from_pretrained(model_name="stt_en_
quartznet15x5") #13
```

#13 asr_model:
`nemo_asr.models.EncDecCTCModel.from_pretrained(model_name="stt_en_quartznet15x5")`
Carga un modelo preentrenado para convertir audio a texto. En este caso, es el modelo QuartzNet para inglés, diseñado para entender y transcribir audio en inglés.

```
# Configuración para convertir Texto a Audio (TTS)
tts_model                                                    =
nemo_tts.models.Tacotron2Model.from_pretrained(model_name="tts_en_
tacotron2") #14
```

#14 tts_model:
`nemo_tts.models.Tacotron2Model.from_pretrained(model_name="tts_en_tacotron2")`
Carga el modelo preentrenado Tacotron2 que convierte texto en una representación visual del sonido llamada espectrograma de mel. Este paso es como preparar el sonido antes de convertirlo en un archivo de audio real.

```
# Consulte los modelos WaveGlow disponibles y seleccione el
adecuado
print(nemo_tts.models.WaveGlowModel.list_available_models()) #15
# Ejemplo: Elija el primer modelo disponible
vocoder_model_name                                              =
nemo_tts.models.WaveGlowModel.list_available_models()[0] #16
vocoder                                                         =
nemo_tts.models.WaveGlowModel.from_pretrained(model_name="tts_en_w
aveglow_88m") #17

# Función para convertir audio a texto
def audio_to_text(audio_file): #18
    if audio_file.startswith('http'): #19 Manejar URLs
        response = requests.get(audio_file) #20
        with open('temp_audio.wav', 'wb') as f: #21
            f.write(response.content) #22
            audio_file = 'temp_audio.wav' #23 Utilice el archivo
descargado
```

#15. print(nemo_tts.models.WaveGlowModel.list_available_models())

- Esta línea de código muestra una lista de programas llamados WaveGlow que se pueden usar para convertir esas imágenes de sonido o espectrogramas (las "imágenes de sonido" son una representación visual de cómo suena algo. Se llaman espectrogramas de mel, y muestran cómo cambian los diferentes tonos (agudos o graves) en el tiempo. Es como si tomaras el sonido y lo convirtieras en una imagen para poder entenderlo mejor antes de convertirlo en audio. Luego, otro programa toma esa imagen y la convierte de nuevo en sonido que podemos escuchar) en audio real que puedes escuchar.

#16. vocoder_model_name =
nemo_tts.models.WaveGlowModel.list_available_models()[0]

- Selecciona el primer modelo WaveGlow de la lista para convertir los espectrogramas en audio. Con esta línea seleccionamos el primer modelo de la lista. Este modelo se almacenará en la variable vocoder_model_name, que luego será utilizado para cargar el modelo específico. Seleccionar un modelo permite ajustar la conversión de espectrogramas a audio según nuestras necesidades: algunos modelos podrían ser más adecuados para ciertos tipos de voces, acentos, o velocidades de conversión.

#17. vocoder = nemo_tts.models.WaveGlowModel.from_pretrained(model_name="tts_en_waveglow_88m")

- Aquí cargamos un modelo preentrenado llamado "tts_en_waveglow_88m". Un modelo preentrenado ha sido entrenado con grandes cantidades de datos para aprender a hacer su tarea, que en este caso es convertir espectrogramas en audio de voz humana. Al cargar este modelo, lo estamos preparando para que haga la conversión de espectrograma a audio.

#18. def audio_to_text(audio_file):

- Define la función que convertirá un archivo de audio en texto. Una función es un bloque de código que realiza alguna operación, en nuestro caso, convertir un archivo de audio en texto.

#19. if audio_file.startswith('http'):

- Verifica si el archivo de audio proviene de una URL. He utilizado esta línea de código debido a que mí archivo de audio proviene de una URL y con la ausencia de esta línea de código el programa no puede exportar el audio y por lo tanto nos dará un error en la salida.

#20. response = requests.get(audio_file)

- Descargo el archivo de audio desde la URL.

#21-23. with open('temp_audio.wav', 'wb') as f: f.write(response.content) audio_file = 'temp_audio.wav'

- Guarda el archivo de audio descargado en el sistema o en mi caso en el entorno de Google Colab como 'temp_audio.wav' para su posterior procesamiento.

```
    # Cargar el audio
    audio, sample_rate = torchaudio.load(audio_file) #24
    # Asegurarse de que el audio esté en la frecuencia correcta
    if sample_rate != 16000: #25
        transform =
torchaudio.transforms.Resample(orig_freq=sample_rate,
new_freq=16000)
        audio = transform(audio) #26-27
```

```
    # Realizar la transcripción
    # El método de transcripción espera una lista de rutas de
archivos, no tensores de audio.
    transcription = asr_model.transcribe([audio_file]) #28
    return transcription[0] #29

# Función para convertir texto a audio
def text_to_audio(text, output_file="output.wav"): #30
    # Generar mel-spectrograma a partir del texto
    parsed = tts_model.parse(text) #31
    spectrogram = tts_model.generate_spectrogram(tokens=parsed)
#32

    # Convertir mel-spectrograma a onda sonora
    audio = vocoder.convert_spectrogram_to_audio(spec=spectrogram)
#33

    # Guardar el archivo de audio
    torchaudio.save(output_file, audio.to('cpu'), 22050) #34
    print(f"Audio saved to {output_file}") #35


from IPython.display import Audio #36

# Ejemplo de uso
audio_file =
'https://dldata-public.s3.us-east-2.amazonaws.com/2086-149220-0033
.wav' #37
transcription = audio_to_text(audio_file) #38
print("Transcription:", transcription) #39
text = "This is a test of the text to speech conversion." #40
text_to_audio(text, output_file="output.wav") #41

# Reproduce el audio generado en Colab
Audio("output.wav", autoplay=True) #42
```

#24. audio, sample_rate = torchaudio.load(audio_file)

- Carga el archivo de audio y obtiene los datos del audio y su frecuencia de muestreo (La frecuencia de muestreo es el número de veces por segundo que se toman mediciones de una señal de audio. Se mide en hercios (Hz).

Por ejemplo, si la frecuencia de muestreo es de 16,000 Hz, significa que cada segundo del audio se ha dividido en 16,000 pequeñas partes para capturar los detalles del sonido. En el código, al cargar el archivo de audio, se obtienen dos cosas: los datos del audio (el sonido real) y su frecuencia de muestreo (cuántas veces por segundo se midió el sonido). Esto es importante porque, para algunos modelos, el audio debe tener una frecuencia de muestreo específica para funcionar correctamente).

#25. if sample_rate != 16000:

- En esta línea evaluo si la frecuencia de muestreo del archivo no coincide con los 16 kHz necesarios por el modelo de transcripción ya que el modelo preentrenado de ASR de NeMo asume que la entrada tiene esta frecuencia específica para operar correctamente.

#26-27. transform = torchaudio.transforms.Resample(orig_freq=sample_rate, new_freq=16000) audio = transform(audio)

- Convierte el audio a 16,000 Hz si la frecuencia de muestreo no coincide. Uso “torchaudio.transforms.Resample” para crear un objeto que cambia la frecuencia de muestreo de un audio. Este objeto necesita dos valores: la frecuencia original del audio (“sample_rate”) y la nueva frecuencia que queremos usar (en mi caso 16 kHz). Después, con “audio = transform(audio)”, se aplica esa transformación al audio cargado, lo que genera un nuevo audio con la frecuencia ajustada a 16 kHz.

#28. transcription = asr_model.transcribe([audio_file])

- El modelo ASR (asr_model) se utiliza para realizar la transcripción del archivo de audio a texto. La función “transcribe” requiere una lista con rutas a los archivos de audio. Por ello, pasamos una lista que contiene únicamente audio_file. El modelo procesa el archivo y me devuelve una lista de transcripciones.

#29. return transcription[0]

- Me devuelve la primera transcripción obtenida del modelo.

#30. def text_to_audio(text, output_file="output.wav"):

- Defino una función que convierte texto en un archivo de audio. Recibe dos parámetros: “text”: una cadena que contiene el texto a convertir en voz.“output_file”: es el nombre del archivo en el que se guardará el audio generado. En mi caso lo he llamado "output.wav".

#31. parsed = tts_model.parse(text)

- Utilizo el modelo de síntesis de voz (`tts_model`) para procesar el texto. El método `parse` convierte el texto en una secuencia de tokens. Estos tokens son partes más pequeñas que el modelo puede procesar.

#32. spectrogram = tts_model.generate_spectrogram(tokens=parsed)

- A partir de los tokens generados en el paso anterior, calculo un espectrograma mel. Un espectrograma mel es una representación gráfica del audio en la que se muestra la energía presente en cada frecuencia a lo largo del tiempo. Es un paso intermedio muy importante en nuestro programa, ya que los modelos de vocoder lo utilizan para reconstruir la onda sonora.

#33. audio = vocoder.convert_spectrogram_to_audio(spec=spectrogram)

- Esta parte línea del código toma el espectrograma, que es como una "imagen del sonido", y lo convierte en audio real que puedes escuchar. Para hacer esto, usamos un programa especial llamado vocoder (en nuestro caso, WaveGlow). El vocoder traduce esa imagen de sonido en ondas sonoras que se transforman en un archivo de audio, como una grabación que puedes reproducir.

#34. torchaudio.save(output_file, audio.to('cpu'), 22050)

- El tensor de audio que he generado anteriormente se guarda en un archivo de formato .wav. El método “audio.to('cpu')” asegura que el tensor esté en el dispositivo correcto (CPU) antes de guardarlo, lo cual es necesario si el modelo estaba trabajando en una GPU. La frecuencia de muestreo del archivo guardado se especifica como 22.05 kHz, una frecuencia estándar en archivos de audio.

#35. print(f"Audio saved to {output_file}")

- Imprimo un mensaje confirmando que el archivo de audio se ha guardado correctamente.

#36 from IPython.display import Audio

- Con esta línea importo la clase Audio desde el módulo “IPython.displayW. Esta clase permite reproducir archivos de audio directamente en entornos interactivos como Google Colab o Jupyter Notebook. Es muy útil para escuchar los resultados de la síntesis de voz o cualquier otro archivo de audio sin necesidad de descargarlo.

#37 audio_file =
'https://dldata-public.s3.us-east-2.amazonaws.com/2086-149220-0033.wav'

- Defino una variable llamada “audio_file” que almacena una URL con relación a un archivo de audio en línea. Este archivo será descargado y utilizado como entrada para la función “audio_to_text”. El archivo contiene una muestra de audio que será transcrita a texto utilizando el modelo de reconocimiento de voz (ASR).

#38 transcription = audio_to_text(audio_file)

- Llamo a la función “audio_to_text”, que procesa el archivo de audio especificado en la URL. La función descarga el archivo, lo ajusta si es necesario (por ejemplo, cambiando la frecuencia de muestreo) y luego utiliza el modelo ASR para transcribir el contenido del audio en formato de texto. El resultado de la transcripción se guarda en la variable que he definido como “transcription”.

#39 print("Transcription:", transcription)

Con este comando imprimo en la consola el texto transcrito por el modelo ASR. Permite visualizar el contenido del audio en formato de texto, facilitando su análisis o uso en otros procesos.

#40 text = "This is a test of the text to speech conversion."

En esta línea defino una variable llamada “text” que contiene una cadena de texto en inglés. Este texto será utilizado como entrada para la función de conversión de texto a audio (TTS).

#41 text_to_audio(text, output_file="output.wav")

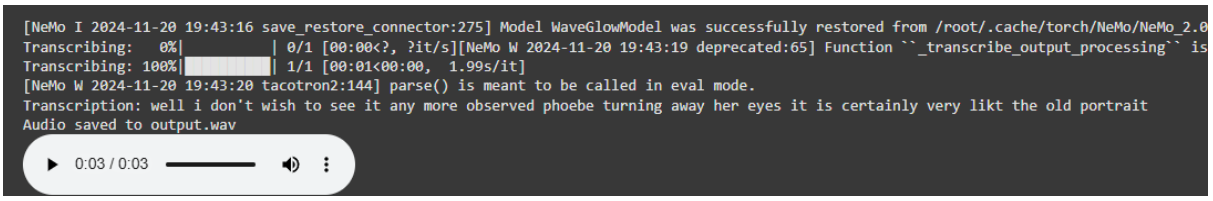
Aquí llamo a la función “text_to_audio” para convertir el texto definido en la línea anterior en un archivo de audio. La función utiliza el modelo TTS para generar un espectrograma mel a partir del texto y luego un vocoder para crear la onda sonora. El audio generado se guarda en un archivo llamado output.wav.

#42 Audio("output.wav", autoplay=True)

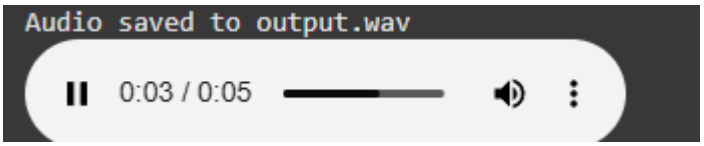
Esta línea usa la clase “Audio” para reproducir el archivo de audio generado (output.wav). El parámetro “autoplay=True” hace que el archivo de audio se reproduzca automáticamente cuando se ejecuta la celda en un entorno interactivo, en mi caso, en Google Colab. Es una forma cómoda de verificar que el texto se ha convertido correctamente en audio y escuchar el resultado directamente desde el notebook.

Una vez finalizado el programa, le damos a ejecutar. Google Colab se conectará a una tarjeta gráfica a través de la nube, en nuestro caso a una Nvidia Tesla K80, una tarjeta gráfica especializada en cálculos complejos, excelente para la IA. El resultado final debe de aparecer abajo del todo, en la consola que se desplegará.

Comenzarán a salir letras y comandos, pero lo más importante está abajo del todo. Para saber si la IA funciona correctamente, nos deberá salir lo siguiente:



En esta captura de pantalla, podemos ver que el proceso de transcripción de audio a texto se ha completado al 100%. Más abajo, nos aparece una línea donde pone “Transcription”, seguida de la transcripción de nuestro archivo de audio. Y por último, el texto que le pasamos en inglés: This is a test of the text to speech conversion. Se nos muestra un reproductor donde nuestro texto es transcrito a audio. Si modificamos el texto a por ejemplo uno más largo, veremos que el audio aumenta de tiempo: This is a test of the text to speech conversion. I hope you enjoy it!



24. Problemas y sus soluciones.

- Problema con la librería Hugging_Face

  El problema que tuve con Hugging_Face creo que está relacionado con varios factores, y lo voy a explicar según lo entendí para que quede claro. Lo primero que se me ocurre es que puede ser por un tema de versiones. Hugging_Face, como muchas librerías, se actualiza con frecuencia, y a veces estas actualizaciones traen cambios que no son compatibles con el código que ya estaba funcionando. Si mi entorno se actualizó automáticamente la versión de la librería “transformers” o alguna otra relacionada, esto podría haber generado un conflicto con el resto de dependencias que uso.

  Otra posibilidad es que haya un conflicto entre las dependencias. Por ejemplo, Hugging_Face necesita trabajar con otras bibliotecas como PyTorch, y si alguna de estas también se actualizó, es posible que ya no sean

compatibles entre sí. Esto puede ser un problema especialmente cuando trabajo en Colab o en un entorno donde no controlo exactamente qué versiones están instaladas o cómo interactúan entre ellas.

También puede ser un problema del entorno donde ejecuto el código. Si estoy usando Colab, sé que ellos actualizan su sistema regularmente, y eso podría haber cambiado las versiones preinstaladas de algunas librerías. A veces esto hace que el código que funcionaba perfectamente deje de hacerlo sin que yo haya cambiado nada.

Algo más que podría estar afectando es cómo se manejan los modelos en Hugging_ Face. Muchas veces descargo modelos directamente de su repositorio, y si en algún momento ellos hacen cambios en esos archivos, como modificar el “config.json”(es un archivo de configuración que muchas librerías y herramientas, como Hugging Face, utilizan para almacenar información clave sobre un modelo o un sistema. En el caso específico de Hugging Face, este archivo contiene los detalles necesarios para definir cómo se comporta un modelo de aprendizaje automático) o el archivo del modelo, el código podría no ser capaz de cargarlos correctamente. Esto pasa sobre todo cuando trabajo directamente desde internet y no tengo los modelos guardados localmente.

Finalmente, si trabajo en Colab, siempre puedo verificar las versiones de las librerías y reinstalar manualmente las que necesito para asegurarme de que todo funciona como antes. En resumen, creo que con más control sobre las versiones y los modelos podría evitar que me vuelva a pasar esto.

- Problema con Google Colab:

  A veces, cuando trabajo en Google Colab, me encuentro con problemas que hacen que el entorno deje de funcionar correctamente, y la única solución es crear uno nuevo. Esto ocurre porque Colab no siempre empieza completamente limpio. Aunque parece que todo se resetea al abrir un nuevo notebook, en realidad puede haber restos de configuraciones, paquetes instalados o cachés que provocan errores. Por ejemplo, si instalo una librería específica que necesita cierta versión de otra dependencia, y en el entorno ya hay instalada una versión diferente, pueden surgir conflictos. Algo similar ocurre con las librerías preinstaladas en Colab, como torch o tensorflow, que a veces no están actualizadas o no son compatibles con los paquetes que intento usar.

  Otro problema común es que, al trabajar durante un tiempo prolongado en un notebook, el entorno acumula variables, datos temporales o configuraciones

que pueden afectar el funcionamiento. También he notado que, si interrumpo una instalación de paquetes o se corta la conexión a mitad de una ejecución, el sistema queda en un estado incompleto. Esto significa que algunas dependencias quedan mal instaladas, y cualquier intento de usar esas librerías puede generar errores inesperados. Además, Colab tiene límites en el uso de recursos como la RAM y la GPU; si los supero o si el entorno lleva demasiado tiempo activo, puede desconectarse o empezar a funcionar de forma errática.

Cuando estos problemas aparecen, crear un nuevo entorno suele ser la mejor solución. Esto se debe a que, al hacerlo, todo se reinicia: las librerías vuelven a sus versiones originales, las configuraciones se limpian y se elimina cualquier error o conflicto que haya surgido en el entorno anterior. Básicamente, es como empezar desde cero en un espacio completamente limpio. Para ello, guardo mi trabajo, cierro el notebook y abro uno nuevo, asegurándome de instalar solo las librerías que realmente necesito y verificando que sean compatibles entre sí. Si alguna instalación previa falló, uso comandos como “!pip install --force-reinstall” para reinstalar desde cero y evitar problemas.

En resumen, los errores en Colab suelen deberse a conflictos de versiones, dependencias mal instaladas o acumulación de configuraciones. Aunque puede ser frustrante, crear un nuevo entorno suele resolver estos problemas al proporcionar un espacio limpio donde todo vuelve a funcionar correctamente. Esto me permite continuar con mi trabajo sin las complicaciones previas.

25. Código Completo

```
!pip install nemo_toolkit['all']
!pip install nemo_text_processing
!pip install pytorch-lightning
!pip install torchaudio
import sys

# Definimos una clase ModelFilter ficticia para satisfacer el requisito de
importación
class ModelFilter:
    pass  # Agregamos cualquier atributo o método necesario si lo requiere
NeMo

# Agregamos el ModelFilter ficticio a sys.modules
sys.modules['huggingface_hub.ModelFilter'] = ModelFilter
```

```
# Ahora podemos importat Nvidia NeMo
import nemo.collections.asr as nemo_asr
import nemo.collections.tts as nemo_tts
import torchaudio
import requests
nemo_asr.models.EncDecCTCModel.list_available_models()
"""
stt_en_quartznet15x5: Este es un modelo QuartzNet que ha demostrado ser muy
eficiente para el reconocimiento de voz en inglés. Es una buena opción si tu
tarea principal es transcribir audio en inglés.

stt_en_jasper10x5dr: Jasper es otra arquitectura de red que se ha utilizado
para ASR. Este modelo es adecuado si necesitas un modelo alternativo al
QuartzNet para la transcripción en inglés.

stt_es_quartznet15x5: Este modelo es específico para español. Si tu audio
está en español, este es el modelo que debes utilizar."""

# Configuración para convertir Audio a Texto (ASR)
asr_model =
nemo_asr.models.EncDecCTCModel.from_pretrained(model_name="stt_en_quartznet1
5x5") #5

# Configuración para convertir Texto a Audio (TTS)
tts_model =
nemo_tts.models.Tacotron2Model.from_pretrained(model_name="tts_en_tacotron2"
) #6

# Consulto los modelos WaveGlow disponibles y seleccione el adecuado
print(nemo_tts.models.WaveGlowModel.list_available_models()) #7
# Ejemplo: Elija el primer modelo disponible
vocoder_model_name =
nemo_tts.models.WaveGlowModel.list_available_models()[0]
vocoder =
nemo_tts.models.WaveGlowModel.from_pretrained(model_name="tts_en_waveglow_88
m")

# Función para convertir audio a texto
def audio_to_text(audio_file):
    if audio_file.startswith('http'): # Handle URLs
        response = requests.get(audio_file)
        with open('temp_audio.wav', 'wb') as f:
            f.write(response.content)
        audio_file = 'temp_audio.wav' # Utilice el archivo descargado

    # Cargar el audio
    audio, sample_rate = torchaudio.load(audio_file)
```

```
    # Asegurarse de que el audio esté en la frecuencia correcta
    if sample_rate != 16000:
        transform = torchaudio.transforms.Resample(orig_freq=sample_rate,
new_freq=16000)
        audio = transform(audio)

    # Realizar la transcripción
    # El método de transcripción espera una lista de rutas de archivos, no
tensores de audio.
    transcription = asr_model.transcribe([audio_file])
    return transcription[0]

# Función para convertir texto a audio
def text_to_audio(text, output_file="output.wav"):
    # Generar mel-spectrograma a partir del texto
    parsed = tts_model.parse(text)
    spectrogram = tts_model.generate_spectrogram(tokens=parsed)

    # Convertir mel-spectrograma a onda sonora
    audio = vocoder.convert_spectrogram_to_audio(spec=spectrogram)

    # Guardar el archivo de audio
    torchaudio.save(output_file, audio.to('cpu'), 22050)
    print(f"Audio saved to {output_file}")

from IPython.display import Audio

# Ejemplo de uso
audio_file =
'https://dldata-public.s3.us-east-2.amazonaws.com/2086-149220-0033.wav'
transcription = audio_to_text(audio_file)
print("Transcription:", transcription)

text = "This is a test of the text to speech conversion. I hope you enjoy
it!"
text_to_audio(text, output_file="output.wav")

# Reproduce el audio generado en Colab
Audio("output.wav", autoplay=True)
```

26. Bibliografía

https://www.youtube.com/watch?v=_L6xPuzg6q0
https://colab.research.google.com/#scrollTo=-Rh3-Vt9Nev9
https://www.nvidia.com/en-gb/ai-data-science/products/nemo/get-started/
https://docs.nvidia.com/nemo-framework/user-guide/latest/nemotoolkit/starthere/intro.html#installation
https://www.youtube.com/watch?v=N4nOsJOKc1k&list=PLYEZllCulzxt-h0jtLkE9-x5z6HNEmHOV&index=2
https://colab.research.google.com/github/NVIDIA/NeMo/blob/stable/tutorials/VoiceSwapSample.ipynb
https://www.youtube.com/watch?v=SwqusllMCnE
https://colab.research.google.com/github/NVIDIA/NeMo/blob/v1.0.0/tutorials/AudioTranslationSample.ipynb
https://medium.com/@khang.pham.exxact/introduction-to-nvidia-nemo-tutorial-example-478f6ba6b160
https://huggingface.co/nvidia/Llama-3.1-Nemotron-70B-Instruct-HF
https://huggingface.co/models?search=Nvidia%20Nemo
https://www.reddit.com/r/learnmachinelearning/comments/15pwrx6/how_to_load_nvidia_nemo_model/?tl=es-es