UNIVERSIDAD
DE ANTIOQUIA

Modelo de Lenguaje

Demóstenes

UNIVERSIDAD
DE ANTIOQUIA

1 8 0 3

Natalia Echeverri
Juan Pablo Montenegro
Santiago Londoño

28 de Enero del 2024

Índice

1. Introducción

1.1. Objetivo general

Desarrollar un modelo de lenguaje capaz de demostrar teoremas matemáticos, integrando el asistente de pruebas formales Lean con técnicas de machine learning.

1.2. Objetivos Específicos

1. **Implementar una API para la integración de Lean con Python.**
   - Establecer una comunicación eficiente entre Lean y Python utilizando Lean Dojo.
   - Automatizar la ejecución de tácticas, la obtención de estados de prueba y la recolección de datos de demostraciones matemáticas.
2. **Construir una base de datos de teoremas demostrables en Lean usando Lean Dojo.**
   - Extraer, estructurar y almacenar información relevante de teoremas, incluyendo estados de prueba, tácticas aplicadas, hipótesis y resultados intermedios.
   - Asegurar que la base de datos contenga metadatos útiles para el entrenamiento del modelo, como el historial de tácticas, los contextos matemáticos y los resultados de pruebas exitosas o fallidas.
3. **Diseñar la arquitectura del modelo de lenguaje para la resolución de demostraciones matemáticas.**
   - Definir la estructura del modelo de aprendizaje automático, considerando arquitecturas de modelos de lenguaje (como transformers o variantes especializadas en tareas de razonamiento lógico).
   - Integrar representaciones de Lean, permitiendo que el modelo comprenda tanto el lenguaje matemático simbólico como las tácticas de Lean.
   - Establecer un *pipeline* de preprocesamiento de datos para convertir el contenido de un script o proyecto Lean en un formato adecuado para el entrenamiento del modelo.
4. **Entrenar el modelo utilizando técnicas de aprendizaje supervisado.**
   - Definir conjuntos de entrenamiento, validación y prueba a partir de la base de datos de teoremas recopilada.
   - Implementar algoritmos de aprendizaje supervisado para que el modelo aprenda a predecir secuencias de tácticas basadas en estados de prueba.
   - Evaluar el rendimiento del modelo en tareas de demostración automática, optimizando sus parámetros para mejorar la precisión y la capacidad de generalización.
   - Analizar los resultados obtenidos para identificar fortalezas y debilidades del modelo en distintos tipos de demostraciones matemáticas.

1.3. Resultados Esperados

- Una API funcional que permita la comunicación fluida entre Lean y Python para automatizar la evaluación de pruebas matemáticas.
- Una base de datos robusta con información estructurada sobre teoremas de Lean, que sirva como recurso para el entrenamiento, testeo y validación del modelo de lenguaje.
- Un modelo de lenguaje pre-entrenado capaz de escribir afirmaciones matemáticas en código Lean encaminado a la resolución de demostraciones matemáticas.
- Un modelo de lenguaje capaz de demostrar teoremas matemáticos en Lean.

2. Estado del arte

2.1. Machine Learning

El aprendizaje automático ha revolucionado múltiples campos, incluyendo ramas de la ciencia como la ingeniería, la física, la astronomía, la bilogía, etc. Permitiendo el análisis de grandes volúmenes de datos de manera eficiente. Modelos como las redes neuronales profundas, el aprendizaje por refuerzo y los métodos bayesianos han demostrado ser herramientas clave en la detección de patrones, clasificación y predicción de fenómenos complejos. En particular, arquitecturas como transformers y técnicas de embedding han mejorado significativamente la representación de datos en espacios de alta dimensión, facilitando la extracción de características relevantes en problemas científicos.

2.2. Definición y conceptos básicos

Lean es un lenguaje de programación funcional basado en teoría de tipos dependientes, diseñado para la verificación formal de teoremas. Su capacidad para garantizar corrección matemática rigurosa lo ha convertido en una herramienta fundamental en la automatización de demostraciones y en la integración con inteligencia artificial (IA).

La relación entre Lean y la IA se basa en el uso de modelos de aprendizaje automático, como transformers y métodos de refuerzo, para asistir en la construcción de pruebas formales.

Técnicas de embeddings permiten representar teoremas y tácticas en espacios de alta dimensión, facilitando la búsqueda y sugerencia de pasos en demostraciones automatizadas.

2.3. Algorítmo PPO

PPO se emplea para entrenar agentes que toman decisiones en entornos dinámicos y complejos.

A diferencia de otros algoritmos más complejos, PPO es relativamente más sencillo y computacionalmente eficiente. Esto lo hace adecuado para una variedad de aplicaciones de machine learning donde la eficiencia es crucial

En muchas aplicaciones, PPO ha demostrado superar a otros algoritmos de aprendizaje por refuerzo en términos de rendimiento y estabilidad.

PPO es una herramienta poderosa y versátil en el campo del machine learning, ofreciendo una combinación de eficiencia, estabilidad y adaptabilidad que lo hace ideal para una variedad de aplicaciones.

2.4. Lean

Lean proof assistant proporciona un amplio banco de teoremas con mathlib de tal manera que se proporcionan matemáticamente teoremas y pruebas de manera interactiva y eficiente, facilitando el desarrollo de software correcto y matemáticas formales.

Lean se puede utilizar como un demostrador de teoremas interactivo. Su programación se centra en la definición de tipos y funciones, permitiendo a los desarrolladores enfocarse en el dominio del problema y en la manipulación de datos, en lugar de los detalles de la programación.

2.5. News

Enseñar matemáticas a una inteligencia artificial requiere traducir sus axiomas fundamentales a un lenguaje comprensible por computadoras. Mientras la IA ha superado a los humanos en juegos estratégicos, la verificación de teoremas exige formalizar las matemáticas en sistemas como Lean. Durante años, investigadores y entusiastas han trabajado en esta tarea, permitiendo que programas basados en Lean asistan a matemáticos en la validación de su trabajo.

2.6. Relevancia en diferentes industrias

Lean tiene una amplia relevancia en diversas industrias debido a sus capacidades de verificación formal y su enfoque en la precisión y la mantenibilidad del código.

Lean se utiliza para verificar la corrección del software, asegurando que las implementaciones cumplan con las especificaciones formales, lo que reduce errores y bugs en el código.

En la industria financiera, Lean puede verificar algoritmos y modelos financieros complejos, garantizando la precisión y la adherencia a las regulaciones.

En la ingeniería de sistemas críticos, como la aeroespacial y la automotriz, Lean asegura que los sistemas y los algoritmos cumplen con los estrictos estándares de seguridad y fiabilidad.

Lean se utiliza en la investigación de inteligencia artificial para probar y verificar formalmente algoritmos, garantizando su precisión y comportamiento esperado.

En sistemas de salud, Lean puede verificar software crítico utilizado en dispositivos médicos, asegurando que funcionen de manera segura y efectiva.

2.7. Intersección entre ML y Lean

Lean puede ser utilizado para verificar formalmente modelos de machine learning. Esto significa que se puede demostrar matemáticamente que un modelo cumple con ciertas propiedades y comportamientos esperados, reduciendo la posibilidad de errores y aumentando la confianza en los resultados.

Lean permite la metaprogramación, lo que significa que se pueden escribir programas que generen otros programas. Esto es útil en machine learning para crear pipelines de datos y modelos de forma dinámica y adaptable.

2.8. Ejemplos de implementación

2.9. Beneficios y desafíos

La integración con Python, facilita el prototipado y la experimentación rápida de nuevos algoritmos y modelos.

Podemos aprovechar las bibliotecas y herramientas de Python, como NumPy, pandas y scikit-learn, mientras utilizamos Lean para asegurar la robustez y precisión de los modelos.

La combinación de Lean y Python puede mejorar la confiabilidad del código, ya que Lean proporciona un marco para la prueba y verificación formal, mientras que Python ofrece flexibilidad y facilidad de uso.

3. Lean Dojo

Lean Dojo es una herramienta diseñada para facilitar la interacción programática con el asistente de pruebas formales **Lean**. Actúa como una API que permite automatizar la ejecución de tácticas, la manipulación de teoremas y la recolección de datos sobre demostraciones matemáticas. Lean Dojo fue desarrollado con el objetivo de integrar Lean con lenguajes de programación como Python, facilitando su uso en aplicaciones de inteligencia artificial y aprendizaje automático.

3.1. ¿Qué es Lean Dojo?

Lean Dojo es una interfaz que conecta el lenguaje formal de Lean con entornos de programación más flexibles, como Python. Esto permite automatizar tareas que tradicionalmente requerirían interacción manual en el entorno de Lean, como la prueba de teoremas, la exploración de tácticas y la extracción de información sobre el estado de las demostraciones.

A través de Lean Dojo, es posible acceder a teoremas formales, ejecutar comandos de Lean y analizar el progreso de una demostración, todo desde un entorno de programación externo. Esto resulta fundamental para proyectos que requieren la automatización de pruebas formales o la integración de Lean en flujos de trabajo de aprendizaje automático.

3.2. Propósito de Lean Dojo

El propósito principal de Lean Dojo es proporcionar una plataforma para la automatización de demostraciones matemáticas y el análisis de teoremas formales. Entre sus objetivos se destacan:

- **Automatización de demostraciones:** Permite ejecutar tácticas de Lean de forma programática, lo que facilita la resolución automatizada de teoremas.

- **Extracción de datos:** Facilita la recopilación de información detallada sobre el proceso de demostración, incluyendo el estado de las pruebas, las hipótesis involucradas y las tácticas aplicadas.
- **Integración con modelos de IA:** Proporciona una base para el entrenamiento de modelos de aprendizaje automático que pueden aprender a resolver teoremas de manera autónoma, utilizando datos extraídos de bibliotecas matemáticas formales como `mathlib4`.

3.3. Funcionalidades de Lean Dojo

Lean Dojo ofrece una variedad de funcionalidades que lo convierten en una herramienta poderosa para la automatización de pruebas formales:

- **Interacción con teoremas:** Permite cargar, analizar y manipular teoremas formales directamente desde código en Python.
- **Ejecución de tácticas:** Facilita la ejecución de tácticas de Lean, como `trivial`, `intro`, `simp`, entre otras, para avanzar en la demostración de teoremas.
- **Análisis del estado de la prueba:** Proporciona información detallada sobre el estado actual de una demostración, incluyendo metas pendientes, hipótesis disponibles y el contexto lógico.
- **Soporte para múltiples versiones de Lean:** Lean Dojo puede trabajar con diferentes commits de repositorios de Lean, lo que permite analizar teoremas en distintas versiones de bibliotecas como `mathlib4`.

3.4. Importancia de Lean Dojo en este Proyecto

En el contexto de este proyecto, Lean Dojo cumple un papel fundamental al servir como el **puente entre Lean y Python**, permitiendo la automatización del análisis de teoremas y la recolección de datos esenciales para el entrenamiento del modelo de lenguaje **Demóstenes**. Gracias a Lean Dojo, fue posible:

- Automatizar la extracción de teoremas y sus demostraciones desde `mathlib4`.
- Analizar el estado de las pruebas para obtener información detallada sobre el proceso de resolución de teoremas.
- Integrar estos datos en un flujo de trabajo de aprendizaje automático, facilitando el entrenamiento de un modelo supervisado capaz de predecir secuencias de tácticas para resolver demostraciones matemáticas.

Lean Dojo no solo agiliza el proceso de recopilación de datos, sino que también permite evaluar el rendimiento del modelo entrenado en escenarios de demostración del mundo real, lo que contribuye significativamente al objetivo de automatizar el razonamiento matemático formal.

4. Base de Datos

La base de datos utilizada para el entrenamiento de Demóstenes está basada en la Benchmark proporcionada por el equipo de LeanDojo (Yang, K. LeanDojo Benchmark 4). Esta es una base de datos creada a partir de la librería mathlib para Lean y leandojo para Python, la cual contiene 122517 teoremas de distintas áreas de las matemáticas. El siguiente es el sistema de archivos de la Benchmark.

```
├─corpus.jsonl
├─metadata.json
├─licenses
│ ├─lean4
│ ├─mathlib4
│ ├─doc-gen4
│ ├─aesop
│ ├─ProofWidgets4
│ ├─std4
│ └─README.md
├─random
│ ├─train.json
│ ├─val.json
│ └─test.json
└─novel_premises
  ├─train.json
  ├─val.json
  └─test.json
```

De este destacamos el archivo corpus.jsonl y los archivos en los directorios random y novel_premises. El archivo corpus contiene todas las premisas matemáticas que potencialmente podrá utilizar Demóstenes para generar código Lean. Mientras que en random y novel_premises se encuentran archivos .json que serán usados para el entrenamiento, validación y testeo. La diferencia entre ambos directorios consiste en la forma en que han sido distribuidos los teoremas en cada archivo json. En novel_premises tenemos certeza de que en el archivo test.json y val.json existen teoremas con premisas que el modelo nunca verá al entrenarse con train.json, esto se hace para medir la capacidad del modelo de usar premisas de las que no tiene referencias en el entrenamiento. Sin embargo, usaremos los archivos en el directorio random, en los que los teoremas han sido repartido por simple azar.

Cada teorema en los archivos json puede ser leido como un diccionario con las siguientes *keys*: 'url', 'commit', 'file_path', 'full_name', 'start', 'end', 'traced_tactics'. Las *keys* 'start' y 'end' nos da información sobre donde inicia y finaliza la demostración del teorema en el archivo al que apunta el file_path dentro del repositorio de mathlib en github. 'traced_tactis' nos da información sobre las tacticas y premisas en el archivos corpus.jsonl que fueron usadas en la demostración del teorema.

Toda esta información, junto con la librería lean-dojo, nos proporcionarán los datos necesarios para realizar el entrenamiento de un modelo base o pre-entrenado capaz de escribir código Lean; y posteriormente evaluar las capacidades demostrativas de un modelo post-entrenado con una arquitectura PPO para enfocar el código producido en demostrar el teorema requerido.

5. Desarrollo de la API

Para la comunicación entre **Lean** y **Python**, se desarrolló una API utilizando **Lean Dojo**. Esta API permite evaluar demostraciones escritas en código Lean desde Python. La integración de Lean con Python proporciona la capacidad de interactuar con el sistema de pruebas, lo que resulta fundamental para el entrenamiento y validación del modelo de lenguaje.

5.1. Entorno de Desarrollo

El entorno de desarrollo fue configurado utilizando **Python 3.11** junto con **Lean 4** y **Lean Dojo**. La elección de Python 3.11 se debió a ser la versión más actualizada de Python compatible con la librería Lean-Dojo. Además, se utilizó Lean 4 debido a su capacidad mejorada para la formalización de matemáticas y la eficiencia en la ejecución de tácticas.

5.2. Integración entre Lean y Python

Lean Dojo fue utilizado como el puente entre Lean y Python. Esta herramienta permite ejecutar comandos y tácticas de Lean directamente desde Python, lo que permite evaluar la demostración de un teorema. La API desarrollada se basó en las siguientes funcionalidades principales:

- **Carga dinámica de teoremas:** Permite seleccionar teoremas específicos dentro del repositorio de Lean para su análisis.
- **Ejecución de tácticas automatizadas:** Posibilita la aplicación de tácticas como `trivial`, `intros` y `simp` de manera programática, sin necesidad de intervención manual.
- **Obtención del estado de la prueba:** Recupera información sobre el estado actual de un teorema, incluyendo hipótesis, metas pendientes y el contexto lógico en el que se desarrolla la demostración.
- **Uso de commits dinámicos:** La API extrae el **hash del commit** desde un archivo JSON, permitiendo trabajar con distintas versiones del repositorio de `mathlib4`, lo que facilita el análisis de teoremas en diferentes estados de desarrollo.

5.3. Arquitectura de la API

La arquitectura de la API sigue un modelo modular, permitiendo la interacción eficiente entre Python y Lean. Se diseñaron distintos módulos que cumplen funciones específicas para garantizar la escalabilidad y el mantenimiento del sistema. A continuación, se describen los componentes principales:

- **Módulo de conexión con Lean:** Responsable de establecer la comunicación con el repositorio de `mathlib4`, utilizando clases como `LeanGitRepo` y `Theorem` para acceder a los teoremas y sus metadatos.
- **Módulo de ejecución de tácticas:** Implementa la lógica para ejecutar tácticas de Lean de forma automatizada, utilizando el objeto `Dojo` para interactuar con el entorno de prueba de Lean.
- **Módulo de extracción de datos:** Encargado de recolectar información sobre el estado de los teoremas, incluyendo el historial de tácticas aplicadas, los resultados de las pruebas y el contexto lógico de cada demostración. Esta información se almacena en una base de datos estructurada para su posterior análisis.

La API fue diseñada para ser flexible, permitiendo la adaptación a diferentes configuraciones de repositorios de Lean y facilitando la incorporación de nuevas funcionalidades en el futuro.

5.4. Resultados y Funcionamiento

La API desarrollada ha permitido la automatización de la interacción con Lean, lo que facilita la extracción de datos sobre teoremas y estados de prueba. Se han logrado los siguientes resultados:

- Extracción de estados de prueba para múltiples teoremas en `mathlib4`, lo que ha permitido obtener información detallada sobre el proceso de demostración.
- Implementación de un flujo de trabajo automatizado para ejecutar tácticas de Lean desde Python, reduciendo significativamente el tiempo necesario para analizar y resolver teoremas de forma manual.
- Integración con una base de datos estructurada que almacena información relevante sobre teoremas, tácticas aplicadas y resultados de pruebas, facilitando su uso en el entrenamiento del modelo de lenguaje.

6. Desarrollo de la arquitectura del modelo

El desarrollo del modelo de lenguaje, denominado **Demóstenes**, se basó en una arquitectura inspirada en el enfoque presentado en el artículo *Attention is All You Need*, que dio origen a modelos de lenguaje de gran éxito como GPT-2 y DeepSeek-Math. El objetivo principal fue entrenar un modelo capaz de asistir en la resolución de demostraciones matemáticas utilizando datos extraídos de `mathlib4` a través de Lean Dojo.

6.1. Estructura de la Implementación

El desarrollo del modelo se organizó en cuatro archivos principales de Python, cada uno con una funcionalidad específica en el flujo de trabajo del proyecto:

1. **Preprocesamiento de Datos:** En este archivo se realizó la limpieza del **dataset de Lean Dojo**, conocido como `Benchmark`. A partir de este conjunto de datos se generaron cuatro archivos JSON:
   - Un archivo que contiene premisas de `mathlib4`, incluyendo tácticas y teoremas que Demóstenes puede utilizar para demostrar otros teoremas.
   - Tres archivos adicionales, cada uno con una lista de diccionarios. Cada diccionario almacena información detallada de un teorema de `mathlib4`, incluyendo:
     - Nombre del teorema.
     - Ruta al archivo `.lean` donde se encuentra en el repositorio de `mathlib4`.
     - Enunciado del teorema junto con su demostración formal en Lean.
2. **Tokenización de Datos:** En este archivo se implementó y entrenó el **tokenizador de Demóstenes** utilizando la librería `tokenizers` de Hugging Face. El proceso de tokenización se dividió en dos fases, cada una correspondiente a un conjunto de datos diferente:
   - La primera fase se centró en el dataset de premisas y tácticas.
   - La segunda fase abarcó los datasets que contienen los enunciados de los teoremas junto con sus demostraciones.

3. **Entrenamiento del Modelo:** En este archivo se llevó a cabo el entrenamiento de Demóstenes utilizando la biblioteca `PyTorch`. Se implementó una arquitectura basada en **transformers**, siguiendo el diseño propuesto en *Attention is All You Need*. El modelo fue entrenado con el dataset de teoremas mediante técnicas de aprendizaje supervisado.
4. **Inferencia del Modelo:** En este archivo se desarrolló el proceso de **inferencia de Demóstenes**. Se carga el archivo `.pth` que contiene los pesos entrenados del modelo, junto con los hiperparámetros utilizados durante el entrenamiento. Además, se incluyen las clases y funciones necesarias para la inferencia, permitiendo que Demóstenes genere demostraciones a partir de nuevos enunciados de teoremas. Este módulo es fundamental para evaluar la capacidad del modelo de aplicar el conocimiento aprendido en contextos no vistos durante el entrenamiento.

6.2. Arquitectura del Modelo

El modelo de lenguaje Demóstenes está basado en una arquitectura de **transformer**, ampliamente utilizada en modelos de lenguaje natural debido a su capacidad para capturar dependencias a largo plazo y relaciones complejas en los datos.

- **Capa de Embedding:** Transforma los tokens de entrada en vectores densos de dimensión fija, facilitando la representación numérica del texto formal.
- **Mecanismo de Atención Multi-Cabeza:** Utiliza 4 cabezas de atención para identificar patrones relevantes en los enunciados y demostraciones de teoremas.
- **Capas del Transformer:** Se implementaron 2 capas de transformer, cada una compuesta por un bloque de atención y una red feed-forward.
- **Redes Feed-Forward:** Cada capa incluye una red feed-forward con una dimensión oculta de 512, que permite la transformación no lineal de las representaciones intermedias.
- **Capa de Salida:** Produce la secuencia de tokens de salida, que corresponde a las tácticas o pasos de demostración predichos por el modelo.

6.3. Configuración del Entrenamiento

El entrenamiento del modelo se realizó utilizando la función de pérdida de **Cross-Entropy**, que es adecuada para tareas de clasificación de secuencias. Se utilizó el optimizador **Adam** debido a su eficacia en la optimización de modelos de deep learning.

Hiperparámetros del Entrenamiento

- **Tamaño del lote (batch size):** 16 teoremas (enunciado con demostración).
- **Épocas:** 4 iteraciones completas sobre el conjunto de entrenamiento.
- **Learning rate:** 0.0005, para controlar la velocidad de actualización de los pesos del modelo.
- **Dimensión del espacio de embeddings:** 512, para representar cada token en un espacio vectorial de alta dimensión.
- **Número de cabezas de atención:** 4, lo que permite al modelo enfocarse en diferentes partes de la secuencia simultáneamente.
- **Número de capas del transformer:** 2, para equilibrar la complejidad del modelo y la eficiencia computacional.
- **Número de parámetros de la capa oculta de cada red feed-forward:** 512, lo que permite una capacidad de representación adecuada.
- **Ventana de contexto:** 256 tokens, para definir el número máximo de tokens que el modelo puede considerar en cada paso de inferencia.

6.4. Proceso de Inferencia

El proceso de inferencia se llevó a cabo utilizando el archivo de pesos entrenados del modelo (`.pth`). Este archivo almacena la configuración de los parámetros aprendidos durante el entrenamiento, que son esenciales para que el modelo pueda generalizar su conocimiento a nuevos teoremas.

Durante la inferencia, se cargan los siguientes elementos:

- **Pesos del modelo:** Para garantizar que la inferencia se realice con los parámetros aprendidos.
- **Hiperparámetros:** Configuraciones clave del modelo, como la dimensión de los embeddings, el número de capas y la ventana de contexto.
- **Clases y funciones de apoyo:** Utilizadas para preprocesar la entrada, manejar el tokenizador y generar la secuencia de salida correspondiente a la demostración predicha por el modelo.

Este módulo de inferencia es crucial para evaluar el rendimiento del modelo en tareas del mundo real, permitiendo verificar su capacidad para generar demostraciones matemáticas de forma autónoma.

6.5. Impacto de la Arquitectura en la Resolución de Demostraciones

La arquitectura del modelo de Demóstenes permitió capturar de manera efectiva la estructura lógica de las demostraciones matemáticas. Gracias al mecanismo de atención y la representación contextual de los tokens, el modelo fue capaz de aprender patrones complejos en la secuencia de tácticas utilizadas en Lean.

Además, la combinación del preprocesamiento de datos, la tokenización eficiente, el diseño del modelo basado en transformers y la capacidad de inferencia resultó fundamental para mejorar la precisión del modelo en la predicción de pasos de demostración, contribuyendo significativamente al objetivo de automatizar el proceso de resolución de teoremas.

7. Resultados y Conclusiones

A continuación mostramos la función de perdida del modelo base y un ejemplo de lo que logra generar dada una conjetura. La función de perdida es evaluada por lotes, conteniendo cada lote 10 teoremas.

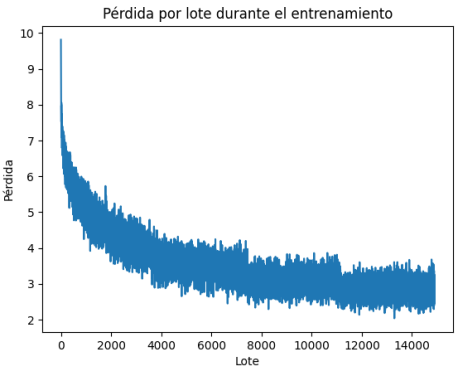

Fig 1: Perdida por lote

```
Modelo cargado desde: demostenes_prueba.pth

Generación usando genera_continuacion:

=== Continuación generada ===
theorem union_comm {α : Type _} (A B : Set α) : A ∪ B
= B ∪ A := by
ext x
simp only [mem_iUnion, and_congr_iff]
rfl
exact hμ

[Finished in 5.4s]
```

Fig 2: Ejemplo de lo generado por el modelo base

- Basándonos en los resultados obtenidos a partir de la función de pérdida, el modelo de lenguaje desarrollado tiene la capacidad de generar código Lean de forma efectiva.
- Se ha demostrado que es factible entrenar un modelo base incluso utilizando hardware con recursos limitados, lo que destaca la eficiencia del enfoque adoptado.
- Finalmente, este modelo está preparado para ser sometido a un proceso de post-entrenamiento, cuyo objetivo será orientar de manera más precisa al modelo hacia la demostración de teoremas específicos.

8. Referencias

1. Yang, K. (2024). LeanDojo Benchmark 4 (Versión v10) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.12740403
2. Yang, K., Swope, A., Gu, A., Chalamala, R., Song, P., Yu, S., ... Anandkumar, A. (2024). Leandojo: Theorem proving with retrieval-augmented language models. Advances in Neural Information Processing Systems, 36.

Respecto al estado del arte, tenemos los siguientes URL dirigidos a papers publicados en Arxviv.

8.1. Artículos de investigación

1. The Lean 4 Theorem Prover and Programming Language
2. A Lean Dataset for International Math Olympiad: Small Steps towards Writing Math Proofs for Hard Problems
3. Formalization of physics index notation in Lean 4
4. Pantograph: A Machine-to-Machine Interaction Interface for Advanced Theorem Proving, High Level Reasoning, and Data Extraction in Lean 4
5. Herald: A Natural Language Annotated Lean 4 Dataset
6. Mathematical Formalized Problem Solving and Theorem Proving in Different Fields in Lean 4
7. LEAN-GitHub: Compiling GitHub LEAN repositories for a versatile LEAN prover
8. Lean Workbook: A large-scale Lean problem set formalized from natural language math problems
9. Process-Driven Autoformalization in Lean 4
10. Formalizing Automated Market Makers in the Lean 4 Theorem Prover

8.2. Libros

1. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition
2. Python for Data Analysis, 3rd Edition
3. Deep Learning with Python
4. Designing Machine Learning Systems
5. Introduction to Proof Through Number Theory
6. Introducción a la lógica matemática - Diego Alejandro Mejía Guzmán.
7. Teoría de conjuntos - Diego Alejandro Mejía Guzmán.
8. Elements of Topology. Puede ser útil la sección: *Chapter 4: Convergence, 4.2 Nets*
9. How to Prove It: A Structured Approach
10. LaTeX Graphics with TikZ: A practitioner's guide to drawing 2D and 3D images, diagrams, charts, and plots
11. Tikz & PGF - Manual for Version 3.1.10

8.3. News

1. Lean - Microsoft Research
2. LEAN-GitHub: A Large-Scale Dataset for Advancing Automated Theorem Proving
3. DeepSeek-AI Open-Sources DeepSeek-Prover-V1.5: A Language Model with 7 Billion Parameters that Outperforms all Open-Source Models in Formal Theorem Proving in Lean 4
4. Can Computers Be Mathematicians?
5. Building the Mathematical Library of the Future

8.4. Adicionales

1. A slightly longer Lean 4 proof tour — What's new - Terence Tao
2. Seminar: Introduction to the Lean 4 theorem prover and programming language by Leonardo de Moura
3. Lean Theorem Prover - ArchLinux
4. **Alpha - Google Deep Mind**
   a) **AlphaProof:** AI achieves silver-medal standard solving International Mathematical Olympiad problems
   b) **AlphaGeometry:** AlphaGeometry: An Olympiad-level AI system for geometry
   c) **AlphaZero:** AlphaZero: Shedding new light on chess, shogi, and Go
   d) **AlphaCode:** Competitive programming with AlphaCode