Universidad de la Habana
Facultad de Matemática y Computación

AutoML con modelos preentrenados

Autor
Daniel Alejandro Cárdenas Cabrera
Tutores
Dr. Yudivian Almeida Cruz

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencia de la Computación

7 de enero de 2023
https://github.com/DanielUH2019/thesis-implementation

Agradecimientos

Muchas veces pensé que no podría llegar a este momento, ha sido un camino muy difícil y lleno de retrocesos. Si estoy aquí es gracias a todas las personas que me han acompañado y motivado a superarme. En especial Lia, Javier y Wendy. Lia, me has dado los mejores momentos que tenido en la facultad y has estado en los más duros, no podía haber pedido mejor pareja para todas esas noches de estrés y trabajo con los proyectos. Javier, desde que nos conocimos nos la hemos pasado hablando de cosas con las que inventar, o como nos gusta llamarle, inflar, y lo más gracioso es que muchas veces nos salió bien, gracias por darle esa emoción a todos estos años. Wendy, mi mejor amiga desde el pre, siempre he podido contar contigo y tu apoyo emocional ha sido fundamental para mí, ya no nos vemos mucho pero todo lo que hemos pasado es involvidable. A Isa y Ramón gracias por la ayuda con cosas de diseño y las muchas risas que nos hemos echado. A Dayan, compañero fiel del gym, dejé de hacer ejercicios muchas veces pero cada vez que volvía podía contar contigo, siempre recordaré todas las fiestas que cogimos juntos y las tardes de dominó. Karina, más conocida como «la ñoñi» y la más fiestera, con lo pequeña que eres no se como podías tener tanta energía, gracias por todo. A Daniel Ernesto, mi compañero en todo el servicio, primero tuve que superar eso para poder llegar a la universidad. A Jaime, Andy, Kevin, Manuel, Miguel, Pili, Morgado y muchos más. A mi familia que siempre me ha apoyado. A todos mis profes, en especial a Yudivian, Piad, Suilan, Daniel, Gian, Celia, Idania, Bruno, Juan Pablo, Somoza y Aymee.

Opinión del tutor

El estudiante Daniel Alejandro Cárdenas Cabrera desarrolló satisfactoriamente el trabajo de diploma titulado "AutoML con modelos preentrenados". En este trabajo el estudiante propone un modelo de AutoML que intenta encontrar una buena combinación de modelos ya preentrenados con el fin de lograr una solución a un problema cualquiera. La solución estará basada en las ideas de AutoGOAL que es la propuesta y la herramienta de AutoML desarrollada por el Grupo de Inteligencia Artificial.

El sistema de AutoML que propone la investigación tiene a los LLMs como un elemento principal. La propuesta toma de base al framework de AutoML, AutoGOAL, y al *framework* DSPy, que permite resolver tareas avanzadas con LLMs y modelos de recuperación, unificando técnicas para prompting y *fine-tuning* de modelos de lenguaje. Además. se hacen unas pruebas de concepto utilizando los datasets HotpotQA, Gsm8k e IMDb.

Este desarrollo continúa la línea de investigación relacionada con AutoML en la que por algunos año ha venido trabajando el Grupo de Investigación en Inteligencia Artificial,

Para poder afrontar el trabajo, el estudiante tuvo que revisar literatura científica relacionada con la temática así como soluciones existentes y bibliotecas de software que pueden ser apropiadas para su utilización. Todo ello con sentido crítico, determinando las mejores aproximaciones y también las dificultades que presentan.

Todo el trabajo fue realizado por el estudiante con una elevada constancia, capacidad de trabajo y habilidades, tanto de gestión, como de desarrollo y de investigación.

Por estas razones pedimos que le sea otorgada al estudiante Daniel Alejandro Cárdenas Cabrera la máxima calificación y, de esta manera, pueda obtener el título de Licenciado en Ciencia de la Computación.

Dr. Yudivián Almeida Cruz

Resumen

El rápido desarrollo de la inteligencia artifial requiere de herramientas cada vez más flexibles que puedan aprovechar los recursos y tiempo invertidos en el entrenamiento de modelos de gran escala, en particular los grandes modelos de lenguaje (LLMs). Estos modelos recientemente han sorprendido a la comunidad científica y las personas en general por los logros alcanzados. El aprendizaje de máquina automático (AutoML) por sus capacidades de optimización para encontrar las mejores soluciones a problemas de aprendizaje de máquina es un campo ideal para el aprovechamiento de modelos preeentrenados. En esta investigación se propone e implementa un sistema de AutoML donde los LLMs son un componentes principal. Se realiza la evaluación del sistema en los *datasets* HotpotQA, Gsm8k e IMDb. El sistema propuesto tiene sus bases en el *framework* de AutoML AutoGOAL y el *framework* DSPy.

Abstract

The rapid development of artificial intelligence requires increasingly flexible tools that can leverage the resources and time invested in training large-scale models, particularly large language models (LLMs). These models have recently astonished the scientific community and the general public with their achievements. Automated machine learning (AutoML), due to its optimization capabilities in finding the best solutions to machine learning problems, is an ideal field for harnessing pretrained models. This research proposes and implements an AutoML system where LLMs are a key component. The system is evaluated on the HotpotQA, Gsm8k, and IMDb datasets. The proposed system is based on the AutoML framework AutoGOAL and the DSPy framework.

Índice

Introducción

En los últimos años la inteligencia artificial ha tenido un rápido y fructífero desarrollo, dotando a la comunidad científica de nuevas técnicas y modelos de aprendizaje [1]. Los mejores resultados en problemas que utilizan conjuntos de datos de imágenes y textos se han conseguido entrenando modelos de aprendizaje profundo (DL) en grandes conjuntos de datos [2], algo al alcance de pocos [3].

Estos modelos logran abordar con gran precisión conjuntos de tareas similares a aquellas para las que fueron originalmente entrenados mediante técnicas de *fine-tuning*[1] [2], lo que permite aprovechar de manera significativa los recursos computacionales utilizados durante la fase inicial de entrenamiento del modelo [4]. A estos modelos se les llama modelos preentrenados. Entre los más exitosos están los Grandes Modelos de Lenguage(LLMs)[2] [5]. Estos modelos almacenan diferentes formas de conocimiento de sentido común a través de varios dominios, alcanzando muy buenos resultados en múltiples tareas sin requerir entrenamiento extra [6]. Además, brindan la capacidad de componer múltiples modelos a través de lenguaje natural, posibilitando así nuevas aplicaciones [7].

El Aprendizaje de Máquina Automático (AutoML) surge como respuesta a la necesidad de automatizar el proceso de construcción de modelos de aprendizaje [1]. Existen varias definiciones de AutoML. Por ejemplo, según Zöller y Huber [8], AutoML está diseñado para reducir la necesidad de científicos de datos y permitir que los expertos en el dominio construyan automáticamente aplicaciones de aprendizaje automático sin mucha necesidad de conocimientos estadísticos o de aprendizaje automático. En «*Taking Human out of Learning Applications: A Survey on Automated Machine Learning*» [9], AutoML se define como una combinación de automatización y aprendizaje automático. En resumen, AutoML se puede entender como la construcción automatizada de un flujo de trabajo de aprendizaje automático dentro de un presupuesto computacional limitado.

[1]Se refiere al proceso de ajustar un modelo preentrenado en un conjunto de datos adicional y específico para adaptarse a una tarea particular.

[2]Típicamente se refieren a modelos de lenguaje que contienen cientos de millones de parámetros y son entrenados con grandes conjuntos de datos de texto

Actualmente existen varias herramientas de AutoML robustas que consiguen resultados que rivalizan e, incluso, superan en algunos casos a los modelos construidos por expertos [10]. Estas han sido efectivas para hacer que el aprendizaje automático sea más accesible mediante la creación de abstracciones de alto nivel que simplifican el proceso de entrenamiento e implementación de modelos [11]. Sin embargo, estas herramientas han destacado principalmente en tareas que utilizan datos tabulares [10], en las cuales no existen métodos eficientes y universalmente reconocidos para reutilizar el conocimiento de los modelos preentrenados [12]. A pesar de esto, AutoGluon [13] consigue utilizar modelos preentrenados en la solución de problemas tabulares multimodales[3] [14] obteniendo buenos resultados, lo que demuestra el potencial de su uso en este tipo de problemas. Son pocas las herramientas de AutoML que pueden resolver problemas puramente de imágenes y textos, como AutoGOAL [10] y AutoKeras [15]. En el caso de AutoKeras se pueden utilizar modelos preentrenados en el espacio de búsqueda [15] pero no está bien documentado su uso, factibilidad y funcionamiento.

Se hace necesario que las herramientas de AutoML tengan una mayor adaptabilidad a diversos dominios y conjuntos de datos, todo ello con el objetivo de incorporar nuevas técnicas de maneras más sencillas, para resolver un mayor número de problemas. Teniendo en cuenta esto, resulta atractivo que las herramientas de AutoML puedan aprovechar los modelos preentrenados. La incorporación de estos modelos preentrenados en las herramientas de AutoML trae consigo desafíos como:

- Incorporar los modelos preentrenados al espacio de búsqueda
- Aprovechar al máximo los recursos que se utilizaron para entrenar los modelos preentrenados

Motivación y Justificación

Con lo visto anteriormente se puede notar que incorporar modelos preentrenados a herramientas de AutoML se presenta como un gran reto para la comunidad científica. Entrenar modelos de aprendizaje de máquina desde cero, en grandes conjuntos datos, con el objetivo de obtener resultados capaces de rivalizar con el estado del arte en el campo, es algo al alcance de pocos, debido a la gran cantidad de recursos computacionales que se requieren. Para que las herramientas de AutoML puedan aplicar los últimos avances en el campo deben ser capaces

[3]Multimodal implica trabajar con datos que provienen de diferentes fuentes o modalidades, como imágenes, texto, audio, video.

de utilizar modelos preentrenados de manera transparente. Esto traería consigo una serie de beneficios adicionales:

- Aprovechar las diversas capacidades de los modelos preentrenados para combinarlos y resolver problemas complejos que un solo modelo por si solo no puede resolver.
- Evaluar y establecer comparaciones de manera sencilla entre varios modelos preentrenados para un mismo problema y seleccionar el que mejor se adapte a las necesidades del usuario.
- Disminuir el impacto ambiental que conllevan los experimentos de aprendizaje de máquina
- Aprovechar plataformas como HuggingFace[4] donde constantemente se publican nuevos modelos preentrenados y que son utilizados ampliamente por desarrolladores, investigadores y entusiastas, contribuyendo a la democratización del aprendizaje de máquina.

Antecedentes

En la facultad de Matemática y Computación de la Universidad de La Habana, el grupo de Inteligencia Artifcial ha dirigido su atención al campo del AutoML y ha desarrollado una herramienta de AutoML hetérogeneo, denominada AutoGOAL [10]. El núcleo de la biblioteca AutoGOAL es la API de bajo nivel que permite a los usuarios proporcionar sus propias implementaciones de algoritmos de aprendizaje automático; declarar los elementos que se deben optimizar (por ejemplo, hiperparámetros); y definir cómo se pueden conectar en flujos complejos de varios pasos. AutoGOAL proporciona un lenguaje simple para definir una gramática que describe el espacio de la solución. En esta investigación se considera que este enfoque es lo suficientemente flexible y poderoso para la incorporación de modelos preentrenados.

Problemática

En la actualidad, las propuestas de AutoML producen modelos para la solución de un problema determinado que debe ser entrenado posteriormente para la solución de ese problema, para ello utilizan el conocimiento que se tiene sobre los tipos de entrada y salida de cada posible algoritmo. En estos momentos se han creado múltiples modelos, algunos con miles de millones de parámetros, que

[4]Hugging Face es una plataforma y comunidad de aprendizaje automático (ML) y ciencia de datos que ayuda a los usuarios a construir, implementar y entrenar modelos de aprendizaje automático. https://huggingface.co/

resuelven un conjunto de tareas particulares. Al momento de esta investigación las herramientas de AutoML que incorporan dichos modelos preentrenados en su espacio de búsqueda son escasas y no está bien documentado su uso. Además, se considera que estas no aprovechan todas las capacidades que ofrecen estos modelos.

Hipótesis de Partida

Teniendo en cuenta todo lo anteriormente analizado, se establece la siguiente hipótesis de partida: es posible incorporar modelos preentrenados en una herramienta de AutoML y permitir que se combinen de manera transparente para resolver un problema determinado.

Objetivos

A partir de la hipótesis planteada, se establece el siguiente Objetivo General para el presente trabajo de investigación:

- Proponer un modelo de AutoML que intente encontrar una buena combinación de modelos ya preentrenados con el fin de lograr una solución a un problema cualquiera. Para esta solución se tomarán ideas de la herramienta de AutoGOAL.

Para alcanzar el cumplimiento del objetivo general es necesario cumplir con un conjunto de objetivos parciales que se detallan a continuación:

- Identificar diferentes modelos preentrenados y herramientas de AutoML presentes en la literatura.
- Analizar AutoGOAL como plataforma de AutoML para incorporar modelos preentrenados.
- Diseñar un espacio de búsqueda que pueda utilizar modelos preentrenados y combinarlos de manera automática para resolver un problema complejo.
- Seleccionar un conjunto de modelos preentrenados que sean efectivos en tareas de diversa índole para integrar al espacio de búsqueda.
- Realizar un prototipo de implementación dentro de AutoGOAL con el espacio de búsqueda propuesto.
- Evaluar la solución propuesta contra otras herramientas de AutoML.

Estructura de la tesis

El resto del documento se encuentra organizado de la siguiente manera:

- Capítulo 1 Se hace un recuento sobre los más importantes avances que han tenido los modelos de aprendizaje automático entrenados en grandes volúmenes de datos y además se estudian diferentes herramientas de AutoML y como algunas han incorporado modelos preentrenados a sus espacios de búsqueda conformando de esta manera el estado del arte.
- Capítulo 2 Se describe la propuesta para utilizar AutoML con modelos preentrenados
- Capítulo 3 Se presentan los detalles de implementación y retos computacionales afrontados, así como la discusión y comparación de los resultados obtenidos.
- Capítulo 4 Se presentan conclusiones y recomendaciones de la investigación, asi como también las referencias bibliográficas y anexos utilizados.

Capítulo 1

1. Estado del Arte

En este capítulo se abordará el inicio y auge de grandes modelos preentrenados asi como sus principales retos y resultados. Dentro de los modelos preentrenados se prestará especial atención a los grandes modelos de lenguage. Además, se hará un breve recorrido por el surgimiento y evolución del AutoML, sus logros, deficiencias y herramientas más prominentes del campo. Esta exploración sentará las bases de esta investigación para la integración de modelos preentrenados a herramientas de AutoML.

1.1. Modelos Preentrenados

Con el lanzamiento de AlexNet [16] se propusieron una serie de modelos de aprendizaje profundo en la comunidad de inteligencia artificial. Estos modelos son capaces de ajustarse mejor a datos complejos en comparación con los modelos convencionales de aprendizaje automático. Desde la perspectiva de su desarrollo (LeNet [17] $\rightarrow$ AlexNet [16] $\rightarrow$ VGG [18] $\rightarrow$ ResNet [19] $\rightarrow$ DenseNet [20]), se puede observar que sus arquitecturas se vuelven cada vez más profundas, y, como la creación de estas arquitecturas es un proceso muy propenso a errores que requiere conocimiento experto y muchas horas de experimentación sus costos de desarrollo son cada vez más elevados [1].

A pesar del éxito de las redes neuronales profundas, varios estudios han descubierto que uno de sus desafíos críticos es la necesidad de contar con grandes cantidades de datos anotados para su entrenamiento [21]. Dada esta problemática, durante el desarrollo de las redes neuronales profundas, se han dedicado esfuerzos masivos a la construcción manual de conjuntos de datos de alta calidad para tareas de inteligencia artificial (Deng et al., 2009 [22]; Lin et al., 2014 [23]; Bojar et al., 2014 [24]), lo que posibilita el aprendizaje de modelos neuronales efectivos para tareas específicas que superan a los modelos no neuronales convencionales [21].

Sin embargo, resulta costoso y requiere mucho tiempo realizar la anotación manual de datos a gran escala. Para abordar este problema, Ashish et al [25]

proponen la red *Transformer*, la cual logra un nuevo rendimiento en el Estado del Arte (SOTA, por sus siglas en inglés) en tareas de traducción automática utilizando *Self-supervised learning* (SSL)[5] para no depender de datos anotados. Posteriormente, el preentrenamiento auto-supervisado en un corpus a gran escala, seguido del *finetuning*[6] en tareas específicas, ha captado la atención de investigadores.

Muchos modelos preentrenados a gran escala han surgido siguiendo este paradigma, como BERT [26], GPT [27], T5 [28], XLNet [29] para procesamiento de lenguage natural, ViT [30], Swin-Transformer [31] en el área de visión por computadora y Whisper [32] para la transcripción de audio. Sin embargo, una limitación importante de este enfoque es que, a menudo, es impracticable hacer *finetuning* con un modelo muy grande(por ejemplo más de 10B de parámetros). Para esto Brown et al (2020) [33] propone un paradigma que permite a los modelos de lenguaje aprender tareas con solo unos pocos ejemplos en forma de demostración, a este paradigma le denomina *In-context Learning* (ICL).

Este paradigma además de no requerir *finetuning*, brinda una interfaz interpretable en lenguaje natural con la que comunicarse con un modelo de lenguaje, siendo mucho más fácil incorporar nuevo conocimiento simplemente cambiando las demostraciones del *prompt*[7] [34]. Esto, unido al surgimiento de varias técnicas de *prompting* como Chain of Thought(COT) [35], ReAct [36] y Program of Thoughts(PoT) [37], ha conseguido que en algunas tareas los modelos a los que se les ha aplicado *finetuning* sean superados por modelos sin entrenamiento extra [35].

Además de *Transformer*, otro tipo de modelos que ha ganado mucha popularidad son los *Diffusion models* [38]. Estos modelos son una familia de modelos generativos probabilísticos [39] que, de manera progresiva, destruyen datos mediante la introducción de ruido, para luego aprender a revertir este proceso con el fin de generar muestras. Han destacado en tareas muy desafiantes como la generación de imágenes [40], audio [41] y videos [42] a partir de instrucciones en lenguaje natural. Sus principales exponentes son StableDiffusion [43], DALL-E 2 [44] y Midjourney[8]

[5]Self-supervised learning (o aprendizaje auto-supervisado en castellano) es una técnica de aprendizaje relativamente reciente donde los datos de entrenamiento se etiquetan de forma autónoma. Los datos se etiquetan encontrando y explotando las relaciones (o correlaciones) entre diferentes inputs al modelo.

[6]Se refiere al proceso de ajustar un modelo preentrenado en un conjunto de datos adicional y específico para adaptarse a una tarea particular.

[7]Instrucción o texto inicial que se le proporciona a una herramienta de IA generativa para guiar su generación de respuestas o resultados

[8]https://www.midjourney.com/

Tal es el auge de la creación de modelos preentrenados que la plataforma HuggingFace cuenta con alrededor de 350 mil modelos entre los que se encuentran los mencionados anteriormente, todo disponible públicamente. Para promover el uso y desarrollo responsable de modelos, los repositorios de modelos están equipados con Tarjetas de Modelo para informar a los usuarios sobre las limitaciones y sesgos de cada modelo. Se puede incluir información adicional, como las tareas, los idiomas y las métricas del modelo. La plataforma permite segmentar los modelos acorde a la tarea que resuelven, teniendo una amplia variedad de tareas a escoger que muestran la existencia de modelos preentrenados para una gran cantidad de dominios. Las tareas son las siguientes:

- Multimodal: texto a imagen, imagen a texto, imagen a video, texto a 3D, etc
- Visión por Computadora: estimación de profundidad, detección de objetos, segmentación de imágenes, etc.
- Procesamiento de Lenguaje Natural: clasificación de texto, similitud de oraciones, generación de texto, traducción, etc.
- Audio: texto a audio, clasificación de audio, detección de voz, audio a audio, etc.
- Tabular: clasificación y regresión
- Aprendizaje por refuerzo.

1.2. AutoML

El AutoML, surge con el objetivo de simplificar y agilizar todo el *pipeline*[9] de aprendizaje automático, buscando una reducción significativa en los costos de desarrollo [1]. Este *pipeline* abarca diversas etapas, comenzando por la preparación de datos, que incluye la recolección y limpieza de datos, abordando aspectos como la identificación y corrección de errores, la gestión de datos faltantes o incoherentes, la eliminación de duplicados y el tratamiento de valores atípicos.

Posteriormente, se realiza la mejora de datos mediante técnicas como la augmentación, que implica entrenar modelos en copias ligeramente modificadas de los datos existentes para mitigar el sobreajuste. La fase de ingeniería de características sigue, comprendiendo la selección, extracción y construcción de características relevantes para el modelo.

La generación del modelo se lleva a cabo en un espacio de búsqueda que define las estructuras del modelo y los métodos de optimización, abarcando desde modelos tradicionales como la Máquina de Vectores Soporte (SVM) [45] hasta Redes Neuronales Profundas (DNN) [46] permitiendo automatizar el di-

[9]Consiste en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo.

seño de sus arquitecturas mediante técnicas como *Neural Architecture Search* (NAS) [47]. Finalmente, la etapa de análisis y validación evalúa la capacidad del modelo para generalizar a datos no vistos, analizando su interpretabilidad y explicabilidad, seguido por el despliegue exitoso en un entorno de producción para realizar predicciones en tiempo real.

Actualmente existen varias herramientas de AutoML, pero la mayoría de ellas solo se centran en el algunas partes del pipeline [1]. Entre las herramientas mas prominentes de la literatura se encuntran: TPOT [48], Hyperopt-Sklearn [49], y Auto-Sklearn [50], construidos sobre scikit-learn [51] para tareas de clasificación y regresión, pero con un enfoque limitado a modelos tradicionales de ML como SVM [45] y KNN [52].

Por otro lado, Auto-Keras [15] se centra en la búsqueda de modelos de aprendizaje profundo utilizando NAS, se especializa en tipos de datos en bruto como imágenes y textos, además de datos estructurados. También es lo suficientemente flexible como para abordar casos de uso de datos multimodales y multitarea. AutoKeras cuenta además con intregración a ClearML [53] , el cual es un sistema completo de código abierto ML / DL gestor de experimentos y solución ML-Ops[10]. Permite a los científicos de datos e ingenieros de datos rastrear, gestionar, comparar y colaborar sin esfuerzo en sus experimentos, así como gestionar fácilmente sus cargas de trabajo de formación en máquinas remotas, esto unido a la integración con *TensorflowCloud*[11] permite correr un modelo entrenado en Google Cloud[12] insertando unas pocas líneas de código extra.

NNI [54], por su parte, destaca como una herramienta robusta y liviana para AutoML, con funciones integradas de ingeniería automática de características, optimización de hiperparámetros y búsqueda de arquitecturas neuronales. Además, NNI puede utilizar algoritmos de otras bibliotecas como scikit-learn [51] y XGBoost [55]. En un enfoque similar, Vega [56] es otra herramienta de AutoML que abarca funciones desacopladas como *Data Augmentation*, optimización de hiperparámetros, búsqueda de arquitecturas neuronales y compresión de modelos.

Autogloun [13], enfocándose en problemas que utilizan datos tabulares consigue automatizar todo el *pipeline* de AutoML antes mencionado, destacando *AutoGluon-Cloud* que tiene como objetivo proporcionar al usuario herramientas para entrenar, ajustar y desplegar modelos respaldados por AutoGluon en la

[10]MLOps es una extensión de la metodología DevOps que busca incluir activos de aprendizaje automático y ciencia de datos como ciudadanos de primera clase dentro de la ecología DevOps.

[11]Es una biblioteca que facilita el entrenamiento y el ajuste de hiperparámetros de modelos Keras en Google Cloud.

[12]Solución de computación en la nube

nube. Con solo unas pocas líneas de código, los usuarios pueden entrenar un modelo y realizar inferencias en la nube sin preocuparse de detalles de *MLOps* como la gestión de recursos.

AutoGOAL permite definir una gramática que describe el espacio de solución [10]. Esto se realiza utilizando un enfoque orientado a objetos donde el usuario define una clase de Python para cada componente de la solución (por ejemplo, cada algoritmo) y anota los parámetros de estas clases con atributos que describen el espacio de valores posibles, que pueden ser tipos básicos (es decir, numéricos, texto, etc.) e instancias de otras clases, recursivamente. En base a las anotaciones, AutoGOAL puede construir automáticamente todas las formas posibles en las que se pueden instanciar las clases del usuario.

En AutoGOAL cada algoritmo debe tener un método run anotado con tipos de datos semánticos (*Document, CategoricalVector, DenseMatrix*, etc) que se utilizan para modelar la compatibilidad entre algoritmos.

El espacio de búsqueda de AutoGOAL se compone de un grafo dirigido y acíclico (DAG) donde cada nodo representa un algoritmo. Las aristas se definen entre algoritmos con tipos de entrada/salida compatibles. Los espacios de configuraciones de hiperparámetros de cada uno de los algoritmos corresponden a gramáticas libres del contexto definidas para cada nodo. AutoGOAL aprovecha esta estructura de grafo definida para un problema específico y genera automáticamente una gramática general a partir del DAG correspondiente.

Actualmente, AutoGOAL ofrece soporte para una amplia variedad de algoritmos de diferentes bibliotecas de aprendizaje como scikit-learn [51], NLTK [57], Tensorflow [58], Spacy [59] y Gensim [60]. Esta herramienta se ha sometido a evaluaciones en diez tareas de aprendizaje de dificultad variable y en distintas fases de desarrollo.

Los resultados demuestran que AutoGOAL es competitivo en tareas tradicionales de aprendizaj supervisado, como la clasificación o la regresión, y sobresale en tareas más difíciles, como el reconocimiento de entidades con nombre, que otros sistemas no pueden manejar [61].

1.3. AutoML con modelos preentrenados

Se han realizado varios experimentos en la incorporación de modelos preentrenados a pipelines de AutoML como en el caso de AutoGluon y AutoKeras. Pero se ha explorado poco el potencial de los grandes modelos de lenguage en esta área. Entre las investigaciones más relevantes está el uso de GPT-4 [62] para realizar NAS [63], permitiendo navegar rápidamente por el espacio de búsqueda de arquitecturas, señalar candidatos prometedores y refinar iterativamente estos candidatos para mejorar el rendimiento. En lugar de apuntar a un rendimiento

de vanguardia, el objetivo es resaltar el potencial de GPT-4 para ayudar en la investigación de un problema técnico desafiante a través de un esquema de indicación simple que requiere un conocimiento de dominio relativamente limitado.

AutoML-GPT [64] sugiere utilizar el lenguaje como interfaz universal para interactuar con los usuarios. Usando LLMs como GPT-4, AutoML-GPT toma dinámicamente las solicitudes de los usuarios desde el modelo y las *model cards* [65], y compone el prompt correspondiente. En última instancia, con este *prompt*, llevará a cabo automáticamente experimentos desde el procesamiento de datos hasta la arquitectura del modelo, la optimización de hiperparámetros y el registro de entrenamiento predicho. Al aprovechar las sólidas capacidades lingüísticas de AutoML-GPT y los modelos de inteligencia artificial disponibles, AutoML-GPT puede abordar numerosas tareas complejas de inteligencia artificial en diversos conjuntos de datos. Este enfoque logra resultados notables en áreas como visión por computadora, procesamiento del lenguaje natural y otras áreas desafiantes.

ZAP (*Zero-Shot* AutoML con modelos preentrenados) [66] combina AutoML, meta-aprendizaje [67] y modelos preentrenados para seleccionar automáticamente el mejor *pipeline* de aprendizaje profundo (modelo preentrenado y *finetuning*) basándose en las características del conjunto de datos. En este contexto se utiliza el término *zero-shot* para expresar que no se realiza ninguna evaluación exploratoria del *pipeline*. ZAP utiliza un gran metaconjunto de datos que contiene el rendimiento de varios *pipelines* de aprendizaje profundo en una amplia gama de conjuntos de datos. Utilizando este metaconjunto de datos, se busca una función que seleccione el *pipeline* de aprendizaje profundo adecuado basándose en las propiedades del conjunto de datos (por ejemplo, la resolución de la imagen y el número de imágenes). Para encontrar esta función se entrena una red neuronal y se consigue alcanzar resultados competitivos en los *AutoDL challenges*[13].

Autogen [68], HuggingGPT [69] y OpenAGI [70] no son herramientas de AutoML pero facilitan el desarrollo y la evaluación de modelos de lenguaje de gran escala en la resolución de tareas complejas y multinivel a través de la manipulación de diversos modelos de expertos en dominio, herramientas, complementos o APIs. Este tipo de tareas no son posibles de resolver con las herramientas de AutoML tradicionales.

Dspy [71] es un *framework* para resolver tareas avanzadas con LLMs y modelos de recuperación (RMs). DSPy unifica técnicas para *prompting* y *finetuning* de modelos de lenguaje, asi como enfoques para el razonamiento, auto-mejora [72] y ampliación con recuperación y herramientas(intérprete de Python, cone-

[13]https://autodl.chalearn.org/

xión con una API para la realización de búsquedas online, etc). Todo esto se expresa a través de módulos que se componen. Para hacer esto posible DSPy proporciona módulos componibles y declarativos para instruir a LLMs mediante una sintaxis Python familiar. Mejora las técnicas de *prompting* como CoT [35] y la *self-reflection* [73], pasando de trucos de manipulación de cadenas adaptadas a mano a operaciones modularmente generalizadas que aprenden a adaptarse a su tarea. DSPy introduce un compilador automático que enseña a los LLMs cómo llevar a cabo los pasos declarativos en su programa. Específicamente, el compilador de DSPy rastreará internamente un programa y luego creará prompts de alta calidad para LLMs (o realizará *finetuning* automáticos para LLMs pequeños) para enseñarles los pasos de su tarea.

1.4. Conclusiones

En lo expuesto anteriormente se puede apreciar un especial interés en el uso del lenguaje natural como interfaz para conectar diversos modelos preentrenados e interactuar con los usuarios de manera más cómoda. Resulta muy prometedora la posibilidad de construir sistemas complejos cuyos componentes son modelos preentrenados y donde las operaciones fundamentales son instrucciones en lenguaje natural. Si se lograra explotar esta idea se pudiera ampliar la participación en el desarrollo de sistemas de inteligencia artificial, prototipar rápidamente sistemas para nuevos dominios y maximizar el valor de componentes especializados preentrenados [74].

Desafortunadamente, se sabe que los LLMs son sensibles a los *prompts* que se les da para cada tarea, y esto se agrava en *pipelines* donde múltiples llamados a LLMs deben interactuar de manera efectiva [71]. Como resultado, los llamados a LLMs en las *pipelines* existentes y en los marcos de desarrollo populares generalmente se implementan mediante cadenas largas de instrucciones y demostraciones que se crean manualmente mediante prueba y error. Khattab et al [71] sostienen que este enfoque, aunque generalizado, puede ser frágil y no escalable, conceptualmente similar a ajustar manualmente los pesos de un clasificador.

En la presente investigación se utilizará DSPy para atacar el problema anterior mediante la creación automática de prompts de alta calidad con los que interactuar con diversos modelos preentrenados. Estableciendo de esta manera un espacio de búsqueda sobre el cual se utilizarán componentes de AutoGOAL para la generación y optimización de pipelines

Capítulo 2

2. Propuesta

En este capítulo se presenta la propuesta para realizar AutoML con modelos preentrenados. Para ello se profundizará en algunos aspectos de DSPy y AutoGOAL, los cuales constituyen los componentes principales de la propuesta.

2.1. DSPy

El compilador de DSPy inicializa *prompts* y *finetuning* a partir de datos mínimos sin necesidad de etiquetas manuales para los pasos intermedios en un programa. Para esto se apoya en los *Signatures*. Un *signature* es una especificación declarativa del comportamiento de entrada/salida de un módulo DSPy. En lugar de invertir esfuerzos en cómo hacer que un LLM realice una sub-tarea, los *signatures* le permiten informar a DSPy cuál es la sub-tarea. Un *signature* consta de tres elementos simples:

- Una descripción mínima de la sub-tarea que se supone que el LLM debe resolver.
- Una descripción de uno o más campos de entrada (por ejemplo, la pregunta de entrada) que se le dará al LLM.
- Una descripción de uno o más campos de salida (por ejemplo, la respuesta a la pregunta) que se esperará del LLM.

2.1.1. Analogía con redes neuronales

Cuando se construyen redes neuronales se utiliza un *framework* como Pytorch [75] para componer capas declaraticas como Convolution [76] o Dropout [77] y entonces se usan optimizadores como SGD [78] o Adam [79] para aprender los párametros de la red. DSPy proporciona módulos de propósito general como CoT [35], PoT [37] o ReAct [36] y se encarga de optimizar sus *prompts* para un programa y una métrica dada. Para esto DSPy cuenta con *Teleprompters*, los cuales son poderosos optimizadores que pueden aprender a iniciar y seleccionar

prompts efectivos para los módulos de cualquier programa. Además, es posible implementar módulos personalizados y expresar cualquier tipo de operaciones dentro del método *forward* correspondiente a cada módulo.

2.2. AutoGOAL

A pesar de que AutoGOAL está diseñado principalmente para AutoML tradicional, puede ser usado en cualquier escenario donde se tienen varias formas posibles de resolver una tarea dada, siempre que el usuario pueda describir a partir de una gramática el espacio de todos los posibles programas que la resuelven [10]. Para esto se puede utilizar la API de bajo nivel de AutoGOAL y su módulo `Grammar` Figura 1, el cual proporciona un conjunto de anotaciones de tipo que se utilizan para definir el espacio de hiperparámetros de una técnica o algoritmo arbitrario. Cada técnica se representa como una clase de Python, y los hiperparámetros correspondientes se representan como argumentos anotados del método `__init__`, ya sea valores primitivos (por ejemplo, numéricos, texto, etc.) o instancias de otras clases, anotadas recursivamente. Dada una colección de clases anotadas, este módulo infiere automáticamente una gramática libre de contexto que describe el espacio de todas las instancias posibles de esas clases. Esta funcionalidad se utiliza para definir los algoritmos que formarán el espacio de búsqueda del sistema propuesto.

El sistema propuesto también se inspira en la idea de los tipos semánticos de AutoGOAL y a cada algoritmo se le define un *signature* de DSPy que permite establecer la compatibilidad mediante descripciones en lenguaje natural de los parámetros de entrada y salida de cada algoritmo. Con esta noción de compatibilidad se construye el grafo que representa el espacio de búsqueda y se reutilizan los algoritmos de búsqueda implementados en AutoGOAL Figura 3.

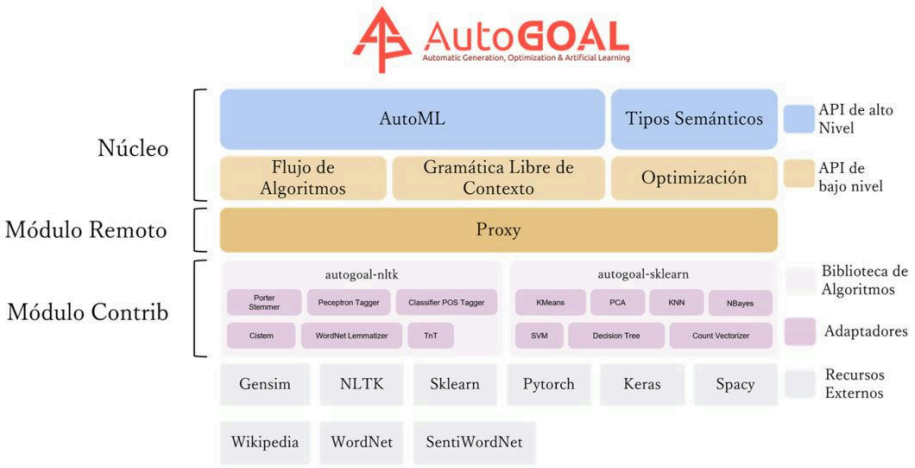

Figura 1: Arquitectura general de AutoGOAL. Tomada de Estevanell [61]

2.3. Propuesta

Se propone un sistema inspirado en la analogía de DSPy con las redes neuronales. Este recibirá una descripción en lenguaje natural de las entradas y salidas que constituyen cada ejemplo de un dataset y una métrica a optimizar. De esta manera se intenta automatizar la creación y composición de módulos de DSPy para resolver un problema dado. A continuación se explican las funcionalidades del sistema:

El sistema cuenta con 6 componentes principales:

1. API de bajo nivel de AutoGOAL para el sampleo y optimización de *pipelines* sobre el espacio de búsqueda creado usando PGE [61] como algoritmo de búsqueda, el cual se encuentra implementado en AutoGOAL.
2. DSPy para la creación automática de *prompts* de alta calidad permitiendo así que un LLM que el usuario escoja como núcleo del sistema pueda enfrentarse a una amplia variedad de problemas que cuenten con algoritmos definidos que usen una de las técnicas de *prompting* de DSPy.
3. Algoritmos que serán los que compongan un *pipeline* sampleado del espacio de búsqueda. Para la definición de un algoritmo es necesario crear un *signature* de DSPy. Luego, la creación de una clase que herede de `DspyAlgorithmBase` a la cual es necesario implementarle un método `get_signature` que devuelve el *signature* previamente definido y un método `run` donde se define como ejecutar ese algoritmo.
4. Una base de datos de vectores [80] como memoria que sirve para mantener datos obtenidos con la ejecución de los algoritmos. Con esto es posible ob-

tener los argumentos necesarios para ejecutar un algoritmo en un momento dado. Además, es utilizada para poder establecer la compatibilidad de un algoritmo con otro a partir de la similitud de sus descripciones en lenguaje natural.

5. Módulo de DSPy para la ejecución de manera secuencial de los algoritmos pertenecientes al *pipeline* correspondiente a través de su método `forward`. Este método puede recibir cualquier cantidad de argumentos asociados a un nombre (`kwargs` en python).
6. Clase `AutoDspy` que actúa como punto de entrada al sistema a través de su método `fit`

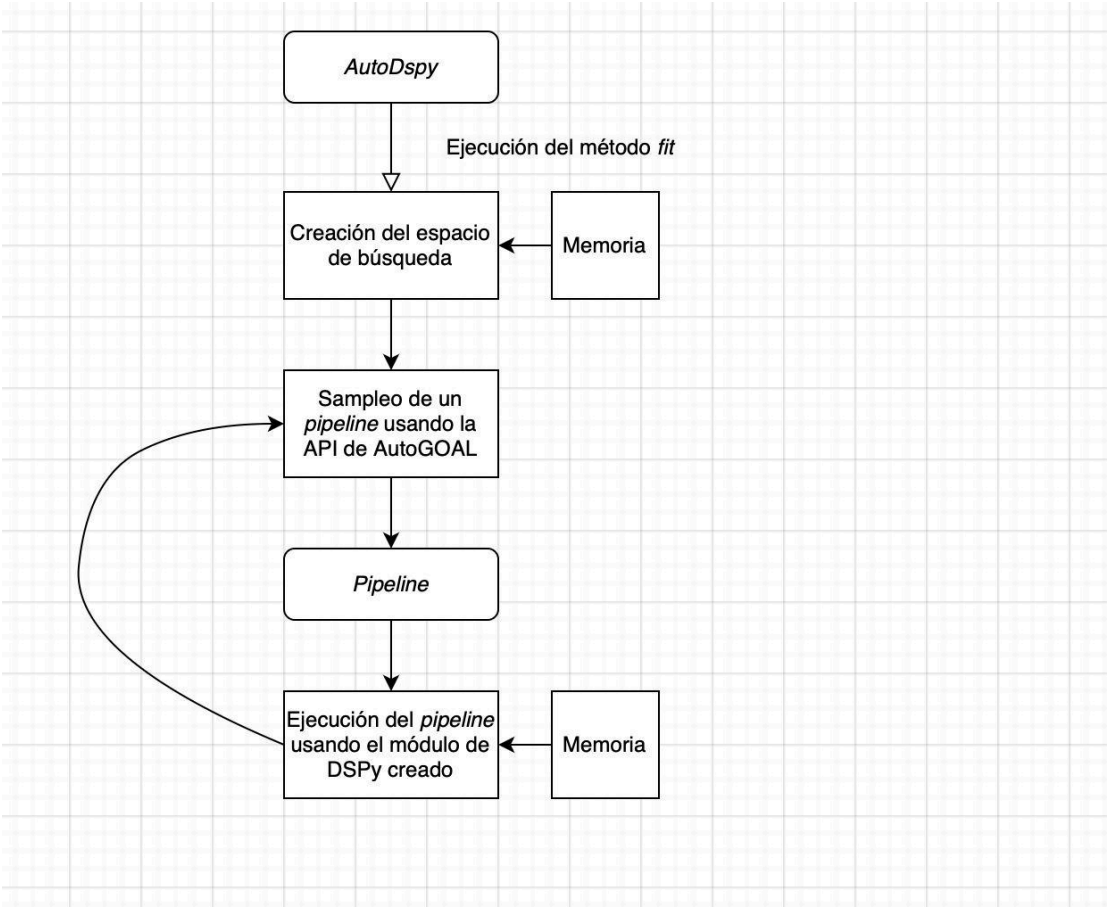

Figura 2: Representación ilustrativa de la interacción de los componentes.

El flujo de trabajo está compuesto por 4 etapas:

1. Preparación de los datos
   - Obtención de un dataset que corresponda a un problema de aprendizaje supervisado[14] con una métrica de evaluación bien definida.

- Crear una descripción en lenguaje natural sobre cada *feature*[15] del dataset y sobre la respuesta.
- A partir del dataset se define una función que constituya la métrica de evaluación.
- Se divide el dataset en conjunto de entrenamiento y validación.
- Se define una función que dada la métrica de evaluación y el conjunto de validación pueda establecer la evaluación de un pipeline encontrado.

2. Instanciación de la clase `AutoDspy`
   - Se selecciona el modelo de lenguaje que será el núcleo del sistema.
   - Se selecciona un algoritmo de búsqueda como PGE.
   - Es posible establecer restricciones a la ejecución como límite de memoria RAM, tiempo máximo de ejecución de la función de evaluación y tiempo máximo de la ejecucón del algoritmo de búsqueda. Esta funcionalidad se encuentra implementada en AutoGOAL y se aprovecha para el sistema.
3. Construcción del espacio de búsqueda
   - Con lo obtenido en la primera etapa se ejecuta el método `fit` de la instancia de la clase de `AutoDspy`
   - Inicialmente se guardan en la base de datos las descripciones de los parámetros de entrada del ejemplo y se intenta inferir cuales pueden ser los posibles algoritmos iniciales compatibles.
   - A partir de cada algoritmo inicial se realiza un DFS[16] que construye las conexiones con el resto de algoritmos simulando su ejecución en una memoria ficticia. Si para cada descripción de un parámetro de entrada de un algoritmo se encuentra en la memoria ficticia un dato con cierto grado de similitud, se considera que un algoritmo es compatible con otro. Esto permite conectar algoritmos que requieran la ejecución previa de 2 o más algoritmos en el *pipeline*.
   - El último algoritmo de cada *pipeline* es un *telepropmter* de DSPy. La manera de que otro algoritmo establezca una conexión con este es dándole a un modelo de lenguage la descripción del algoritmo correspondiente y de la salida del ejemplo del dataset, luego se le pregunta si ese algoritmo podría resolver un problema que requiere esa salida. Se restrinje la respuesta del modelo a verdadero o falso.
   - Un ejemplo de la construcción del espacio de búsqueda sería el siguiente: supongamos que se tiene un problema donde se quiere analizar el sentimiento de críticas de películas. La entrada sería la crítica y su descripción «crítica de una película», su salida sería la clasificación, la cual podría tener como descripción «clasificación de un texto». El sistema podría establecer conexiones con algoritmos como hacer una búsqueda en internet para obtener contexto y de clasificación de texto. Este último sería detec-

tado como válido para dar la respuesta final al problema y por lo tanto se establecería una conexión entre dicho algoritmo y el *teleprompter*.

4. Entrenamiento y validación.
   - Después de creado el espacio de búsqueda, el método `fit` utiliza el algoritmo de búsqueda seleccionado para samplear *pipelines* en ese espacio.
   - Se ejecuta el *teleprompter* perteneciente al pipeline sobre el módulo de DSPy definido por el sistema. Esto devuelve un programa creado por el compilador de DSPy constituyendo de esta manera el entrenamiento.
   - Se ejecuta el programa en el conjunto de validación

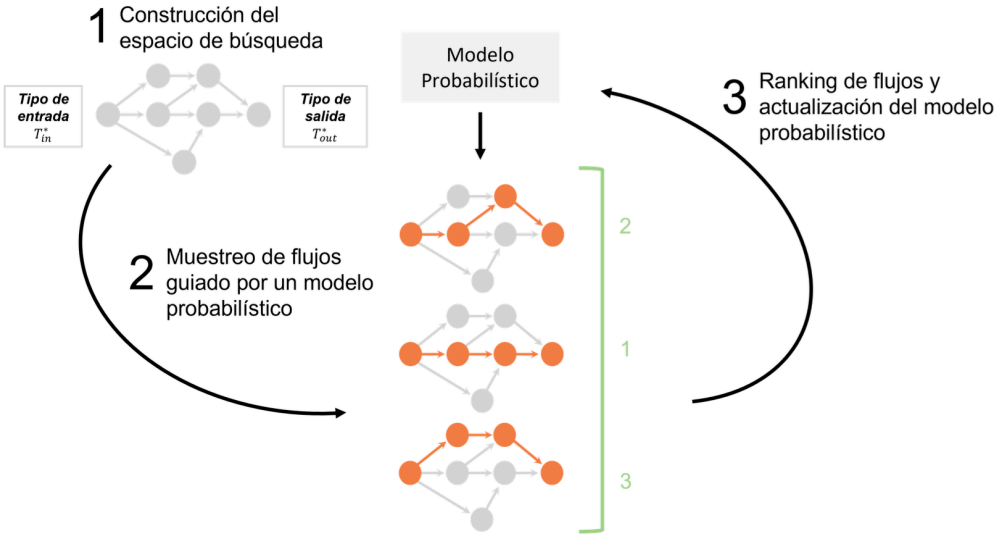

Figura 3: Representación ilustrativa de los pasos del proceso de optimización de AutoGOAL. Tomada de Estevanell [61]

Con esta propuesta se considera que se crea un interesante entorno de investigación para experimentar y explorar una nueva manera de hacer AutoML utilizando modelos de lenguaje como componentes principales. Los cuales pueden contribuir a una integración más sencilla de diferentes modelos preentrenados e incluso herramientas(intérprete de Python, conexión con una API para la realización de búsquedas online, etc) fuera del ámbito del AutoML para resolver problemas que no pueden solucionar herramientas de AutoML tradicionales.

[14]https://es.wikipedia.org/wiki/Aprendizaje_supervisado
[15]Característica o atributo de un conjunto de datos
[16]https://en.wikipedia.org/wiki/Depth-first_search

Capítulo 3

3. Pruebas de Concepto

El objetivo de este capítulo consiste en explicar detalladamente el proceso de experimentación realizado con la implementación del sistema propuesto.

3.1. Experimento

La experimentación realizada tiene como objetivo fundamental mostrar indicios de las potencialidades del sistema para evaluar si sería beneficioso invertir más tiempo y recursos computacionales para su mejora en el futuro. Para esto se decidió realizar pruebas de concepto implementando una pequeña cantidad de algoritmos restringidos a los dominios de generación de texto y análisis de sentimientos. Se seleccionaron tres *datasets*, dos de geneación de texto y uno de análisis de sentimientos priorizando la facilidad de integración con el sistema.

3.2. *Datasets* escogidos

- HotpotQA [81] Conjunto de datos diseñado para la tarea de pregunta y respuesta (QA, por sus siglas en inglés) que se enfoca en la resolución de preguntas que requieren la consulta y el razonamiento sobre múltiples documentos. El conjunto de datos HotpotQA se destaca por sus preguntas que involucran la comprensión y el análisis de información dispersa en varios pasajes de texto para proporcionar respuestas precisas. Se seleccionaron aleatoriamente 20 ejemplos para el conjunto de entrenamiento y 30 para la validación.
- Gsm8k [82] Consiste en una colección de 8.500 problemas matemáticos de escuela primaria, creados por escritores humanos y de alta calidad lingüística. Está segmentado en 7.500 problemas de entrenamiento y 1.000 problemas de prueba. Estos problemas requieren entre 2 y 8 pasos para resolverse, y las soluciones implican principalmente realizar una secuencia de cálculos elementales utilizando operaciones básicas. El dataset se utiliza para evaluar la lógica y las habilidades matemáticas en LLMs, y ha sido utilizado en varios

benchmarks. Se seleccionaron aleatoriamente 30 ejemplos para el conjunto de entrenamiento y 25 para la validación.
- IMDb [83] Conjunto de datos para análisis de sentimientos binarios que consta de 50,000 reseñas de películas de la base de datos IMDb, etiquetadas como positivas o negativas. Se seleccionaron aleatoriamente 30 ejemplos para el entrenamiento y 25 para la validación.

Para la evaluación de los tres *datasets* se utiliza la métrica exact_match que es la evaluación de la coincidencia exacta entre la salida del modelo y la etiqueta esperada.

3.3. Algoritmos implementados

El sistema esencialmente lo que hace es crear dinámicamente un módulo de DSPy para resolver un problema. Debido a esto añadir un algoritmo al espacio de búsqueda requiere de la creación de dos clases. Una clase que se encarga de la definición de su *signature*, es decir, establecer una especificación declarativa del comportamiento de entrada/salida de un módulo DSPy y una clase que se le asigne ese *signature* y contenga la implementación del algoritmo correspondiente. Los signatures de los algoritmos implementados son los siguientes:

```
class GenerateQuestion(AlgorithmSignature):
    """Generate a question from a text that can be used to better understand the
text."""

    text = InputField(desc="the text to be understood")
    question = OutputField(
        desc="a question that can be asked to better understand a text"
    )


class GenerateSearchQuery(AlgorithmSignature):
    """Write and run simple search query that will help answer a complex question."""

    question = InputField(desc="question")
    answer = OutputField(desc="may contain relevant facts")


class GenerateAnswer(AlgorithmSignature):
    """Answer questions with short factoid answers using some relevant facts."""

    context = InputField(desc="may contain relevant facts")
    question = InputField()
    answer = OutputField(desc="often between 1 and 5 words")


class BasicQA(AlgorithmSignature):
```

```
    """Answer questions with short factoid answers."""

    question = InputField()
    answer = OutputField(desc="often between 1 and 5 words")


class SummarizeText(AlgorithmSignature):
    """Generate a concise summary of a given text."""

    text = InputField(desc="the input text to be summarized")
    summary = OutputField(desc="a concise summary of the input text")


class TextClassification(AlgorithmSignature):
    """Analyze the sentiment of a given piece of text and classify it"""

    text = InputField(desc="text for sentiment analysis")
    predicted_classification = OutputField(
          desc="the classification of the input text, e.g., positive, negative,
neutral"
    )


class WikipediaRetrieval(AlgorithmSignature):
    """Retrieve information from wikipedia to provide useful facts"""

    question = InputField(desc="question")
    context = OutputField(desc="may contain relevant facts")
```

Se escogió implementar los siete algoritmos definidos anteriormente porque a pesar de ser simples su combinación pudiera dar lugar a la creación de *pipelines* con cierto grado de complejidad. Un ejemplo de *pipeline* donde se puede aprovechar la combinación de estos algoritmos sería en el análisis de sentimientos de un texto. Para esto pudieran crearse *pipelines* que primero resuman el texto, permitiendo así el uso de modelos que de otra manera no pudieran usarse si el texto original excediera la longitud máxima que son capaces de procesar.

3.4. Modelos preentrenados para cada algoritmo

Todos los algoritmos cuentan con la posibilidad de usar CoT para que el LLM del sistema intente resolver la tarea correspondiente. Además, los algoritmos con los siguientes *signatures* cuentan con la posibilidad de utilizar modelos preentrenados de HuggingFace:

- *GenerateAnswer*
  - *deepset/tinyroberta-squad2*[17]

    - *distilbert-base-cased-distilled-squad*[18]
- *SummarizeText*
    - *Falconsai/text_summarization*[19]
    - *facebook/bart-large-cnn*[20]
- *TextClassification*
    - *cardiffnlp/twitter-roberta-base-sentiment-latest*[21]
    - *distilbert-base-uncased-finetuned-sst-2-english*[22]

Para añadir más modelos preentrenados a los *signatures* anteriores basta con agregar el nombre del modelo nuevo de manera tal que coincida con uno existente en HuggingFace, pueda ser cargado con su biblioteca *transformers* y soporte la tarea correspondiente. No se añadió una mayor cantidad de modelos debido a limitaciones de conectividad.

Ejemplo de como se definen los modelos que puede seleccionar un algoritmo:

```
class GenerateAnswerAlgorithm(DspyAlgorithmBase):
    def __init__(
        self,
        method: CategoricalValue(
            COT_MODULE,
            "deepset/tinyroberta-squad2",
            "distilbert-base-cased-distilled-squad",
        ),
    ):
        self.method = method
```

3.5. *Hardware* y *sofbware*

- Laptop con sistema operativo Nobara Linux 38
- Cpu Ryzen 9 5900hs
- 32GB de RAM
- GPU Nvidia RTX 3060 6GB
- Python 3.10.13
- transformers 4.36.2[23]
- llama-cpp-python 0.2.25[24]
- *Fork* propio de DSPy con soporte añadido para llama-cpp-python[25]
- LLM openhermes-2.5-mistral-7b.Q4_K_M.gguf[26]

[17]https://huggingface.co/deepset/tinyroberta-squad2
[18]https://huggingface.co/distilbert-base-cased-distilled-squad
[19]https://huggingface.co/Falconsai/text_summarization
[20]https://huggingface.co/facebook/bart-large-cnn
[21]https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment-latest
[22]https://huggingface.co/distilbert-base-uncased-finetuned-sst-2-english

- Código fuente de AutoGOAL 1.0[27]

3.6. Restricciones

- Tiempo máximo de ejecución de la función de evaluación: 2 horas
- Tiempo máximo de ejecución del algoritmo de búsqueda: 8 horas
- Cantidad máxima de memoria RAM utilizada: 24GB

3.7. Resultados

3.7.1. HotpotQA

El sistema encontró un total de 19 *pipelines* de los cuales todos pudieron ser evaluados satisfactoriamente y 16 de ellos tuvieron 0% de precisión. La mejor precisión alcanzada fue de 16.67% con los siguientes *pipelines*:

```
algorithms=[
      BasicQAAlgorithm(prompt_technique="ChainOfThought"),
      TeleprompterAlgorithm(),
  ]

algorithms=[
      GenerateAnswerAlgorithm(method="deepset/tinyroberta-squad2"),
      BasicQAAlgorithm(prompt_technique="ChainOfThought"),
      TeleprompterAlgorithm(),
  ]

algorithms=[
      SummarizeTextAlgorithm(method="Falconsai/text_summarization"),
      BasicQAAlgorithm(prompt_technique="ChainOfThought"),
      SummarizeTextAlgorithm(method="ChainOfThought"),
      GenerateAnswerAlgorithm(method="distilbert-base-cased-distilled-squad"),
      BasicQAAlgorithm(prompt_technique="ChainOfThought"),
      TeleprompterAlgorithm(),
  ]
```

El resto de los *pipelines* obtuvo precisión 0, entre ellos se encuentran los siguientes:

```
algorithms=[
      GenerateAnswerAlgorithm(method="deepset/tinyroberta-squad2"),
```

[23]https://huggingface.co/docs/transformers/index
[24]https://github.com/abetlen/llama-cpp-python
[25]https://github.com/DanielUH2019/dspy/tree/llama-cpp
[26]https://huggingface.co/TheBloke/OpenHermes-2.5-Mistral-7B-GGUF
[27]https://github.com/autogoal/autogoal

```
        TeleprompterAlgorithm(),
    ]

algorithms=[
        SummarizeTextAlgorithm(method="facebook/bart-large-cnn"),
        WikipediaRetrievalAlgorithm(),
        TeleprompterAlgorithm(),
    ]

algorithms=[
        WikipediaRetrievalAlgorithm(),
        GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
        SummarizeTextAlgorithm(method="ChainOfThought"),
        BasicQAAlgorithm(prompt_technique="ChainOfThought"),
        GenerateAnswerAlgorithm(method="distilbert-base-cased-distilled-squad"),
        SummarizeTextAlgorithm(method="ChainOfThought"),
        GenerateAnswerAlgorithm(method="deepset/tinyroberta-squad2"),
        TeleprompterAlgorithm(),
    ]

algorithms=[
        WikipediaRetrievalAlgorithm(),
        GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
        TeleprompterAlgorithm(),
    ]

algorithms=[WikipediaRetrievalAlgorithm(), TeleprompterAlgorithm()],

algorithms=[
        GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
        TeleprompterAlgorithm(),
    ]

algorithms=[
        SummarizeTextAlgorithm(method="facebook/bart-large-cnn"),
        TeleprompterAlgorithm(),
    ]
```

3.7.2. Gsm8k

El sistema encontró un total de 12 *pipelines* de los cuales todos pudieron ser evaluados satisfactoriamente y 7 de ellos tuvieron 0% de precisión. La mejor precisición encontrada fue del 44% con el siguiente *pipeline*:

```
algorithms=[
        GenerateAnswerAlgorithm(method="ChainOfThought"),
        TeleprompterAlgorithm(),
    ]
```

Otros con evaluación fueron los siguientes con 32% y 8% respectivamente:

```
algorithms=[
        SummarizeTextAlgorithm(method="facebook/bart-large-cnn"),
        BasicQAAlgorithm(prompt_technique="ChainOfThought"),
        GenerateAnswerAlgorithm(method="ChainOfThought"),
        WikipediaRetrievalAlgorithm(),
        SummarizeTextAlgorithm(method="Falconsai/text_summarization"),
        GenerateAnswerAlgorithm(method="ChainOfThought"),
        TeleprompterAlgorithm(),
    ]
```

```
algorithms=[
        BasicQAAlgorithm(prompt_technique="ChainOfThought"),
        TeleprompterAlgorithm(),
    ]
```

Entre los que tuvieron 0% de precisión se encuentran:

```
algorithms=[WikipediaRetrievalAlgorithm(), TeleprompterAlgorithm()],
```

```
algorithms=[
        GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
        SummarizeTextAlgorithm(method="Falconsai/text_summarization"),
        BasicQAAlgorithm(prompt_technique="ChainOfThought"),
        GenerateAnswerAlgorithm(method="ChainOfThought"),
        WikipediaRetrievalAlgorithm(),
        GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
        WikipediaRetrievalAlgorithm(),
        SummarizeTextAlgorithm(method="ChainOfThought"),
        TeleprompterAlgorithm(),
    ]
```

3.7.3. IMDb

El sistema encontró 9 *pipelines* de los cuales 3 lanzaron excepción al superar las 2 horas de evaluación, 5 tuvieron 0% de precisión y solo uno consiguió superar ese valor alcanzando un 8%.

- Mejor *pipeline*

```
algorithms=[
      SummarizeTextAlgorithm(method="facebook/bart-large-cnn"),
      TextClassificationAlgorithm(method="ChainOfThought"),
      TeleprompterAlgorithm(),
    ]
```

- *Pipelines* que lanzaron excepción

```
algorithms=[
      GenerateAnswerAlgorithm(method="ChainOfThought"),
      TextClassificationAlgorithm(
          method="distilbert-base-uncased-finetuned-sst-2-english"
      ),
      BasicQAAlgorithm(prompt_technique="ChainOfThought"),
```

```
        TextClassificationAlgorithm(
            method="distilbert-base-uncased-finetuned-sst-2-english"
        ),
        BasicQAAlgorithm(prompt_technique="ChainOfThought"),
        TextClassificationAlgorithm(method="ChainOfThought"),
        GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
        TeleprompterAlgorithm(),
    ]
```

```
algorithms=[
        GenerateAnswerAlgorithm(method="deepset/tinyroberta-squad2"),
        GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
        BasicQAAlgorithm(prompt_technique="ChainOfThought"),
        GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
        TextClassificationAlgorithm(
            method="cardiffnlp/twitter-roberta-base-sentiment-latest"
        ),
        SummarizeTextAlgorithm(method="Falconsai/text_summarization"),
        TextClassificationAlgorithm(
            method="cardiffnlp/twitter-roberta-base-sentiment-latest"
        ),
        GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
        TextClassificationAlgorithm(
            method="cardiffnlp/twitter-roberta-base-sentiment-latest"
        ),
        BasicQAAlgorithm(prompt_technique="ChainOfThought"),
        SummarizeTextAlgorithm(method="facebook/bart-large-cnn"),
        BasicQAAlgorithm(prompt_technique="ChainOfThought"),
        SummarizeTextAlgorithm(method="facebook/bart-large-cnn"),
        TeleprompterAlgorithm(),
    ]
```

```
algorithms=[
        SummarizeTextAlgorithm(method="ChainOfThought"),
        TextClassificationAlgorithm(method="ChainOfThought"),
        SummarizeTextAlgorithm(method="ChainOfThought"),
        GenerateAnswerAlgorithm(method="deepset/tinyroberta-squad2"),
        TeleprompterAlgorithm(),
    ]
```

- Ejemplo de *pipelines* que obtienen 0% de precisión

```
algorithms=[
        TextClassificationAlgorithm(
            method="distilbert-base-uncased-finetuned-sst-2-english"
        ),
        WikipediaRetrievalAlgorithm(),
        SummarizeTextAlgorithm(method="ChainOfThought"),
        GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
        GenerateAnswerAlgorithm(method="distilbert-base-cased-distilled-squad"),
        BasicQAAlgorithm(prompt_technique="ChainOfThought"),
        GenerateAnswerAlgorithm(method="deepset/tinyroberta-squad2"),
        TeleprompterAlgorithm(),
    ]
```

```
algorithms=[
      TextClassificationAlgorithm(method="ChainOfThought"),
      WikipediaRetrievalAlgorithm(),
      TeleprompterAlgorithm(),
  ]
```

```
algorithms=[
      TextClassificationAlgorithm(
          method="distilbert-base-uncased-finetuned-sst-2-english"
      ),
      SummarizeTextAlgorithm(method="ChainOfThought"),
      TextClassificationAlgorithm(
          method="distilbert-base-uncased-finetuned-sst-2-english"
      ),
      GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
      TextClassificationAlgorithm(method="ChainOfThought"),
      WikipediaRetrievalAlgorithm(),
      GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
      TextClassificationAlgorithm(
          method="distilbert-base-uncased-finetuned-sst-2-english"
      ),
      GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
      SummarizeTextAlgorithm(method="ChainOfThought"),
      WikipediaRetrievalAlgorithm(),
      GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
      TeleprompterAlgorithm(),
  ]
```

3.8. Análisis de los resultados

No es posible encontrar y evaluar una gran cantidad de *pipelines* en el tiempo dado, por lo que es posible que los recursos computacionales usados sean una limitación importante para el sistema. En los 3 *datasets* los *pipelines* con 0% de precisión representan la mayor cantidad. Se puede apreciar que una de las causas de esto es que la manera en que se establece que un algoritmo puede dar respuesta a un problema y por tanto determina la salida del *pipeline*, no es la adecuada. Esto pasa en *pipelines* como los siguientes:

- HotpotQA

  ```
  algorithms=[
        GenerateQuestionAlgorithm(prompt_technique="ChainOfThought"),
        TeleprompterAlgorithm(),
    ]
  ```

  En este contexto, resulta evidente que un algoritmo diseñado para formular preguntas no podrá resolver un problema que demande una respuesta a una pregunta.

- Gsm8k

  ```
  algorithms=[WikipediaRetrievalAlgorithm(), TeleprompterAlgorithm()]
  ```

  No es posible que un algoritmo que recupera información de Wikipedia pueda resolver un problema de matemáticas
- IMDb

  ```
  algorithms=[
        TextClassificationAlgorithm(method="ChainOfThought"),
        WikipediaRetrievalAlgorithm(),
        TeleprompterAlgorithm(),
    ]
  ```

  Un algoritmo que recupera información de Wikipedia no es capaz de devolver una clasificación positiva o negativa sobre un texto.

Si se toman los mejores *pipelines* encontrados para cada dataset se puede observar que son *pipelines* muy simples y fácilmente implementables en DSPy. No se consigue aprovechar todo el potencial de DSPy, ya que un pequeño módulo como el siguiente, implementado a mano y ejecutado en las mismas condiciones que el sistema propuesto, es capaz de alcanzar un 53.3% de precisión en el dataset HotpotQA:

```
class RAG(dspy.Module):
    def __init__(self, num_passages=3):
        super().__init__()

        self.retrieve = dspy.Retrieve(k=num_passages)
        self.generate_answer = dspy.ChainOfThought(GenerateAnswer)

    def forward(self, question):
        context = self.retrieve(question).passages
        prediction = self.generate_answer(context=context, question=question)
        return dspy.Prediction(context=context, answer=prediction.answer)
```

Se comprobó el rendimiento de los modelos preentrenados de HuggingFace escogidos en los *datasets*. En HotpotQA y Gsm8k los modelos que funcionan en ellos no superan el 0% de precisión. En IMDb los modelos de clasificación de texto usados no son capaces de procesar ninguno de los ejemplos del dataset debido a que superan la longitud máxima que son capaces de procesar. Con esto se determina que la experimentación realizada no fue capaz de mostrar el potencial de usar modelos preentrenados especializados de HuggingFace.

Los resultados no son satisfactorios pero cumplen con los objetivos planteados en la experimentación. Las limitaciones de recursos computacionales y conectividad, junto con la evaluación en tan pocos *datasets* pueden haber tenido un efecto significativo en los resultados.

3.9. Comparación con otras herramientas

AutoML-GPT no ha sido liberado y no fue posible ejecutar ZAP. AutoGOAL y AutoKeras no son capaces de ejecutar los dos primeros *datasets* pero en IMDb son capaces de obtener muy buenos resultados. AutoGOAL con 1 hora de búsqueda y un tiempo máximo de evaluación de 5 minutos es capaz de alcanzar un 85% de precisión. AutoKeras según su sitio web[28] consigue un 93.93% en 1.2 días de GPU usando una GPU NVIDIA Tesla V100.

[28]https://autokeras.com/benchmarks/

Conclusiones

En esta investigación se consiguió proponer e implementar una manera novedosa de hacer AutoML con modelos preentrenados usando las bases de Auto-GOAL. La implementación, a pesar de constituir una aproximación inicial, se pudo validar como prueba de concepto a través de la experimentación realiza. Se determina que los objetivos propuestos al inicio de la investigación fueron cumplidos.

Se destaca como punto fuerte del sistema propuesto la fácil incorporación de más algoritmos y modelos preentrenados de HuggingFace. Además, se considera que usuarios sin mucha experiencia podrían utilizar el sistema, ya que solo necesitan cargar un dataset, brindar descripciones en lenguaje naturar sobre sus ejemplos y separarlo en entrenamiento y validación.

En el sistema propuesto se comprobó que la manera de establecer que algoritmo puede ser el último en el *pipeline*, no consigue buenos resultados. También, debido a el poco tiempo de desarrollo y los limitados recursos computacionales no se consiguió una experimentación más completa con mayor cantidad y variedad de *datasets*, algoritmos y modelos preentrenados.

Recomendaciones

La implementación de la propuesta es solo una prueba de concepto y está en edades muy tempranas de desarrollo. A continuación se propondrán una serie de ideas que se considera podrían mejorar los resultados obtenidos en esta investigación:

1. Definir más algoritmos que enriquezcan el espacio de búsqueda como generación y ejecución de código y que puedan resolver problemas que involucren otras modalidades(audio, imágenes, tabular, multimodal). Sería interesante, además, experimentar con la generación automática de *signatures* usando un LLM.
2. Añadir al espacio de búsqueda la selección de otras técnicas de *prompting* como ReAct, Pot y ToT en aquellos que lo permitan para una mejor explotación del potencial de DSPy y los LLMs.
3. Incorporar una mayor cantidad y variedad de modelos preentrenados de HuggingFace
4. Idear otras maneras de establecer la compatibilidad entre algoritmos en el espacio de búsqueda y en particular como determinar si un algoritmo puede ser el último del pipeline para contribuir a la creación de *pipelines* más robustos.
5. Experimentar con una mayor cantidad de *datasets* que abarquen problemas más diversos para identificar con mejor precisión las potencialidades de la propuesta.
6. Repetir la experimentación contando con mayores recursos computacionales. Permitiendo de esta manera encontrar y evaluar una gran cantidad de *pipelines*-.
7. Explorar el uso de heurísticas para intentar estimar, en cierto momento, cuando el entrenamiento y validación de un *pipeline* excederá los límites de tiempo. De esta manera se podría conseguir un uso más eficiente de las horas de experimentación.
8. DSPy cuenta con varios tipos de *teleprompters* y cada uno tiene diferentes parámetros. Estos podrían definirse como el resto de algoritmos para que AutoGOAL optimice la selección y ajuste de los *teleprompters*. Con esto se

podrían identificar las ventajas de cada *telepropmter* según el problema en que se esté trabajando.

9. Pudiera definirse el LLM como un tipo de algoritmo y hacer que AutoGOAL encuentre el más adecuado entre un conjunto proporcionado por el usuario. Así podrían encontrarse situaciones en que LLMs más pequeñas destaquen sobre las de mayor tamaño, lo que conllevaría a soluciones más eficientes.

Bibliografía

[1] X. He, K. Zhao, y X. Chu, «AutoML: A Survey of the State-of-the-Art», *ArXiv*, 2019, Disponible en: https://api.semanticscholar.org/CorpusID:199405568

[2] K. S. Kalyan, A. Rajasekharan, y S. Sangeetha, «AMMUS : A Survey of Transformer-based Pretrained Models in Natural Language Processing», *arXiv preprint arXiv:2108.05542*, 2021.

[3] M. Artetxe *et al.*, «OPT: Open Pre-trained Transformer Language Models», *arXiv preprint arxiv:2205.01068*, 2022.

[4] T. Dettmers, A. Pagnoni, A. Holtzman, y L. Zettlemoyer, «QLoRA: Efficient Finetuning of Quantized LLMs», *arXiv preprint arXiv:2305.14314*, 2023.

[5] W. X. Zhao *et al.*, «A Survey of Large Language Models», *arXiv preprint arXiv:2303.18223*, 2023, Disponible en: http://arxiv.org/abs/2303.18223

[6] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, y Y. Iwasawa, «Large Language Models are Zero-Shot Reasoners», *arXiv*, 2022.

[7] A. Zeng *et al.*, «Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language», *arXiv*, 2022.

[8] M.-A. Zöller y M. F. Huber, «Benchmark and Survey of Automated Machine Learning Frameworks», *J. Artif. Intell. Res.*, pp. 409–472, 2019, Disponible en: https://api.semanticscholar.org/CorpusID:210064426

[9] Q. Yao *et al.*, «Taking Human out of Learning Applications: A Survey on Automated Machine Learning», 2018. Disponible en: https://api.semanticscholar.org/CorpusID:53109768

[10] SuilanEstevez-Velarde, Y. Gutiérrez, Andres Montoyo, y Y. A. Cruz, «Automatic Discovery of Heterogeneous Machine Learning Pipelines: An Application to Natural Language Processing», 2020, Disponible en: https://aclanthology.org/2020.coling-main.317

[11] D. Xin, E. Y. Wu, D. J. L. Lee, N. Salehi, y A. G. Parameswaran, «Whither AutoML? Understanding the Role of Automation in Machine Learning Workflows», *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, Disponible en: https://api.semanticscholar.org/CorpusID:231592695

[12] K. S. J. H. M. P. Vadim Borisov Tobias Leemann y G. Kasneci, «Deep Neural Networks and Tabular Data: A Survey», *arXiv preprint arXiv: 2110.01889*, 2021.

[13] N. Erickson *et al.*, «AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data», *arXiv preprint arXiv:2003.06505*, 2020.

[14] X. Shi, J. Mueller, N. Erickson, M. Li, y A. Smola, «Multimodal AutoML on Structured Tables with Text Fields», 2021.

[15] H. Jin, F. Chollet, Q. Song, y X. Hu, «AutoKeras: An AutoML Library for Deep Learning», *Journal of Machine Learning Research*, n.º 6, pp. 1–6, 2023, Disponible en: http://jmlr.org/papers/v24/20-1355.html

[16] A. Krizhevsky, I. Sutskever, y G. E. Hinton, «ImageNet classification with deep convolutional neural networks», *Communications of the ACM*, pp. 84–90, 2012, Disponible en: https://api.semanticscholar.org/CorpusID:195908774

[17] Y. LeCun, L. Bottou, Y. Bengio, y P. Haffner, «Gradient-based learning applied to document recognition», *Proc. IEEE*, pp. 2278–2324, 1998, Disponible en: https://api.semanticscholar.org/CorpusID:14542261

[18] K. Simonyan y A. Zisserman, «Very Deep Convolutional Networks for Large-Scale Image Recognition», *CoRR*, 2014, Disponible en: https://api.semanticscholar.org/CorpusID:14124313

[19] K. He, X. Zhang, S. Ren, y J. Sun, «Deep Residual Learning for Image Recognition», *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015, Disponible en: https://api.semanticscholar.org/CorpusID:206594692

[20] G. Huang, Z. Liu, y K. Q. Weinberger, «Densely Connected Convolutional Networks», *CoRR*, 2016, Disponible en: http://arxiv.org/abs/1608.06993

[21] X. Han *et al.*, «Pre-Trained Models: Past, Present and Future», *ArXiv*, 2021, Disponible en: https://api.semanticscholar.org/CorpusID:235421816

[22] X. Chen *et al.*, «Microsoft coco captions: Data collection and evaluation server», *arXiv preprint arXiv:1504.00325*, 2015.

[23] X. Li *et al.*, «Oscar: Object-Semantics Aligned Pre-training for Vision-Language Tasks», 2020. Disponible en: https://api.semanticscholar.org/CorpusID:215754208

[24] O. Bojar *et al.*, «Findings of the 2014 Workshop on Statistical Machine Translation», O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, C. Monz, M. Post, y L. Specia, Eds., Baltimore, Maryland, USA: Association for Computational Linguistics, jun. 2014, pp. 12–58. doi: 10.3115/v1/W14-3302.

[25] A. Vaswani *et al.*, «Attention is All you Need», 2017. Disponible en: https://api.semanticscholar.org/CorpusID:13756489

[26] J. Devlin, M.-W. Chang, K. Lee, y K. Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding», 2019. Disponible en: https://api.semanticscholar.org/CorpusID:52967399

[27] A. Radford y K. Narasimhan, «Improving Language Understanding by Generative Pre-Training», 2018. Disponible en: https://api.semanticscholar.org/CorpusID:49313245

[28] C. Raffel *et al.*, «Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer», *J. Mach. Learn. Res.*, pp. 1–67, 2019, Disponible en: https://api.semanticscholar.org/CorpusID:204838007

[29] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, y Q. V. Le, «XLNet: Generalized Autoregressive Pretraining for Language Understanding», 2019. Disponible en: https://api.semanticscholar.org/CorpusID:195069387

[30] A. Dosovitskiy *et al.*, «An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale», *ArXiv*, 2020, Disponible en: https://api.semanticscholar.org/CorpusID:225039882

[31] Z. Liu *et al.*, «Swin Transformer: Hierarchical Vision Transformer using Shifted Windows», *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9992–10002, 2021, Disponible en: https://api.semanticscholar.org/CorpusID:232352874

[32] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, y I. Sutskever, «Robust Speech Recognition via Large-Scale Weak Supervision», *ArXiv*, 2022, Disponible en: https://api.semanticscholar.org/CorpusID:252923993

[33] T. B. Brown *et al.*, «Language Models are Few-Shot Learners», *ArXiv*, 2020, Disponible en: https://api.semanticscholar.org/CorpusID:218971783

[34] Q. Dong *et al.*, «A Survey on In-context Learning», 2022. Disponible en: https://api.semanticscholar.org/CorpusID:255372865

[35] J. Wei *et al.*, «Chain of Thought Prompting Elicits Reasoning in Large Language Models», *ArXiv*, 2022, Disponible en: https://api.semanticscholar.org/CorpusID:246411621

[36] S. Yao *et al.*, «ReAct: Synergizing Reasoning and Acting in Language Models», *ArXiv*, 2022, Disponible en: https://api.semanticscholar.org/CorpusID:252762395

[37] W. Chen, X. Ma, X. Wang, y W. W. Cohen, «Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks», *ArXiv*, 2022, Disponible en: https://api.semanticscholar.org/CorpusID:253801709

[38] L. Yang *et al.*, «Diffusion Models: A Comprehensive Survey of Methods and Applications», *ACM Computing Surveys*, pp. 1–39, 2022, Disponible en: https://api.semanticscholar.org/CorpusID:252070859

[39] L. Ruthotto y E. Haber, «An introduction to deep generative modeling», *GAMM-Mitteilungen*, 2021, Disponible en: https://api.semanticscholar.org/CorpusID:232168940

[40] C. Zhang, C. Zhang, M. Zhang, y I.-S. Kweon, «Text-to-image Diffusion Models in Generative AI: A Survey», *ArXiv*, 2023, Disponible en: https://api.semanticscholar.org/CorpusID:257505012

[41] C. Zhang *et al.*, «A Survey on Audio Diffusion Models: Text To Speech Synthesis and Enhancement in Generative AI», *ArXiv*, 2023, Disponible en: https://api.semanticscholar.org/CorpusID:257913174

[42] Z. Xing *et al.*, «A Survey on Video Diffusion Models», *ArXiv*, 2023, Disponible en: https://api.semanticscholar.org/CorpusID:264172934

[43] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, y B. Ommer, «High-Resolution Image Synthesis with Latent Diffusion Models». 2021.

[44] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, y M. Chen, «Hierarchical Text-Conditional Image Generation with CLIP Latents», *ArXiv*, 2022, Disponible en: https://api.semanticscholar.org/CorpusID:248097655

[45] I. Steinwart y A. Christmann, «Support vector machines», *Wiley Interdisciplinary Reviews: Computational Statistics*, 2008, Disponible en: https://api.semanticscholar.org/CorpusID:661123

[46] H. Larochelle, Y. Bengio, J. Louradour, y P. Lamblin, «Exploring Strategies for Training Deep Neural Networks», *J. Mach. Learn. Res.*, pp. 1–40, 2009, Disponible en: https://api.semanticscholar.org/CorpusID:996073

[47] B. Zoph y Q. V. Le, «Neural Architecture Search with Reinforcement Learning», *ArXiv*, 2016, Disponible en: https://api.semanticscholar.org/CorpusID:12713052

[48] R. S. Olson y J. H. Moore, «TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning», 2016. Disponible en: https://api.semanticscholar.org/CorpusID:12399099

[49] B. Komer, J. Bergstra, y C. Eliasmith, «Hyperopt-Sklearn: Automatic Hyperparameter Configuration for Scikit-Learn», 2014. Disponible en: https://api.semanticscholar.org/CorpusID:6083252

[50] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, y F. Hutter, «Auto-sklearn: Efficient and Robust Automated Machine Learning», 2019. Disponible en: https://api.semanticscholar.org/CorpusID:160011452

[51] F. Pedregosa *et al.*, «Scikit-learn: Machine Learning in Python», *ArXiv*, 2011, Disponible en: https://api.semanticscholar.org/CorpusID:10659969

[52] L. E. Peterson, «K-nearest neighbor», *Scholarpedia*, p. 1883, 2009, Disponible en: https://api.semanticscholar.org/CorpusID:29611121

[53] ClearML, «ClearML - Your entire MLOps stack in one open-source tool». [En línea]. Disponible en: https://clear.ml/

[54] NNI, «NNI - Neural Network Intelligence». [En línea]. Disponible en: https://nni.readthedocs.io/

[55] T. Chen y C. Guestrin, «XGBoost: A Scalable Tree Boosting System», *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, Disponible en: https://api.semanticscholar.org/CorpusID:4650265

[56] B. Wang *et al.*, «VEGA: Towards an End-to-End Configurable AutoML Pipeline», *ArXiv*, 2020, Disponible en: https://api.semanticscholar.org/CorpusID:226237541

[57] S. Bird, «NLTK: The Natural Language Toolkit», 2006. Disponible en: https://api.semanticscholar.org/CorpusID:1438450

[58] M. Abadi *et al.*, «TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems», *ArXiv*, 2016, Disponible en: https://api.semanticscholar.org/CorpusID:5707386

[59] M. Honnibal y I. Montani, «spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing», 2017.

[60] R.˘Reh̊ůrek y P. Sojka, «Software Framework for Topic Modelling with Large Corpora», Valletta, Malta: ELRA, may 2010, pp. 45–50.

[61] E. Valladares, «AutoML Heterogéneo Multiobjetivo: Una propuesta para la democratización del aprendizaje automático», 2023.

[62] O. J. Achiam *et al.*, «GPT-4 Technical Report», 2023. Disponible en: https://api.semanticscholar.org/CorpusID:257532815

[63] M. Zheng *et al.*, «Can GPT-4 Perform Neural Architecture Search?», *ArXiv*, 2023, Disponible en: https://api.semanticscholar.org/CorpusID:258291577

[64] S. Zhang, C. Gong, L. Wu, X. Liu, y M. Zhou, «AutoML-GPT: Automatic Machine Learning with GPT», *ArXiv*, 2023, Disponible en: https://api.semanticscholar.org/CorpusID:258480269

[65] M. Mitchell *et al.*, «Model Cards for Model Reporting», *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 2018, Disponible en: https://api.semanticscholar.org/CorpusID:52946140

[66] E. Ozturk, F. Ferreira, H. S. Jomaa, L. Schmidt-Thieme, J. Grabocka, y F. Hutter, «Zero-Shot AutoML with Pretrained Models», 2022. Disponible en: https://api.semanticscholar.org/CorpusID:249847968

[67] P. Brazdil, J. N. van Rijn, C. Soares, y J. Vanschoren, «Metalearning: Applications to Automated Machine Learning and Data Mining», *Metalearning*, 2022, Disponible en: https://api.semanticscholar.org/CorpusID:247078592

[68] Q. Wu *et al.*, «AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework», *ArXiv*, 2023, Disponible en: https://api.semanticscholar.org/CorpusID:260925901

[69] Y. Shen, K. Song, X. Tan, D. S. Li, W. Lu, y Y. T. Zhuang, «HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face», *ArXiv*, 2023, Disponible en: https://api.semanticscholar.org/CorpusID:257833781

[70] Y. Ge, W. Hua, J. Ji, J. Tan, S. Xu, y Y. Zhang, «OpenAGI: When LLM Meets Domain Experts», *ArXiv*, 2023, Disponible en: https://api.semanticscholar.org/CorpusID:258049306

[71] O. Khattab *et al.*, «DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines», *arXiv preprint arXiv:2310.03714*, 2023.

[72] J. Huang *et al.*, «Large Language Models Can Self-Improve», *ArXiv*, 2022, Disponible en: https://api.semanticscholar.org/CorpusID:253080328

[73] N. Shinn, F. Cassano, B. Labash, A. Gopinath, K. Narasimhan, y S. Yao, «Reflexion: Language Agents with Verbal Reinforcement Learning», 2023. Disponible en: https://api.semanticscholar.org/CorpusID:258833055

[74] O. Khattab *et al.*, «Demonstrate-Search-Predict: Composing Retrieval and Language Models for Knowledge-Intensive NLP», *arXiv preprint arXiv: 2212.14024*, 2022.

[75] A. Paszke *et al.*, «PyTorch: An Imperative Style, High-Performance Deep Learning Library», 2019. Disponible en: https://api.semanticscholar.org/CorpusID:202786778

[76] Y. LeCun y Y. Bengio, «Convolutional networks for images, speech, and time series», 1998. Disponible en: https://api.semanticscholar.org/CorpusID:6916627

[77] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, y R. Salakhutdinov, «Dropout: a simple way to prevent neural networks from overfitting», *J. Mach. Learn. Res.*, pp. 1929–1958, 2014, Disponible en: https://api.semanticscholar.org/CorpusID:6844431

[78] S. Ruder, «An overview of gradient descent optimization algorithms», *ArXiv*, 2016, Disponible en: https://api.semanticscholar.org/CorpusID:17485266

[79] D. P. Kingma y J. Ba, «Adam: A Method for Stochastic Optimization», *CoRR*, 2014, Disponible en: https://api.semanticscholar.org/CorpusID:6628106

[80] Y. Han, C. Liu, y P. Wang, «A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge», *ArXiv*, 2023, Disponible en: https://api.semanticscholar.org/CorpusID:264289073

[81] Z. Yang *et al.*, «HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering», 2018. Disponible en: https://api.semanticscholar.org/CorpusID:52822214

[82] K. Cobbe *et al.*, «Training Verifiers to Solve Math Word Problems», *ArXiv*, 2021, Disponible en: https://api.semanticscholar.org/CorpusID:239998651

[83] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Ng, y C. Potts, «Learning Word Vectors for Sentiment Analysis», 2011. Disponible en: https://api.semanticscholar.org/CorpusID:1428702