

Textbooks are all you need

Suriya Gunasekar Yi Zhang Jyoti Aneja Caio C'esar Teodoro Mendes
Allie Del Giorno Sivakanth Gopi Mojan JavaheripiPiero Kauffmann
Gustavo de RosaOlli Saarikivi Adil Salim Shital ShahHarkirat Singh Behl
Xin Wang Sebastien BubeckRonen Eldan Adam Tauman Kalai Yin Tat Lee
Yuanzhi Li

Investigación de Microsoft

Resumen

Presentamos **phi-1**, un nuevo modelo de gran lenguaje para código, con un tamaño significativamente menor que los modelos de la competencia: **phi-1** es un modelo basado en Transformer con 1,3B parámetros, entrenado durante 4 días de 8 A100, utilizando una selección de datos de "textbook quality" de la web (6B tokens) y textbooks y ejercicios generados sintéticamente con GPT-3.5 (1B tokens). A pesar de esta pequeña escala, **phi-1** alcanza una precisión **pass@1** del 50,6% en HumanEval y del 55,5% en MBPP. También muestra propiedades emergentes sorprendentes en comparación con **phi-1-base**, nuestro modelo *antes de* la fase de ajuste en un dataset de ejercicios de codificación, y **phi-1-small**, un modelo más pequeño con 350 millones de parámetros entrenado con el mismo proceso que **phi-1**, que aún alcanza el 45% en HumanEval.

1 Introducción

El arte de entrenar grandes redes neuronales artificiales ha progresado extraordinariamente en la última década, especialmente tras el descubrimiento de la arquitectura Transformer [VSP⁺ 17], aunque la ciencia que hay detrás de este éxito sigue siendo limitada. En medio de una vasta y confusa serie de resultados, surgió una apariencia de orden más o menos al mismo tiempo que se introdujeron los Transformers, a saber, que el rendimiento mejora de forma predecible a medida que se aumenta la cantidad de cálculo o el tamaño de la red [HNA⁺ 17], un fenómeno que ahora se conoce como *leyes de escala* [KMH⁺ 20]. La posterior exploración de la escala en el aprendizaje profundo se guio por estas leyes de escalado [BMR⁺ 20], y los descubrimientos de variantes de estas leyes condujeron a un rápido salto en los rendimientos [HBM⁺ 22]. En este trabajo, siguiendo los pasos de Eldan y Li [EL23], exploramos la mejora que puede obtenerse a lo largo de un eje diferente: la *calidad* de los datos. Desde hace tiempo se sabe que unos datos de mayor calidad conducen a mejores resultados; por ejemplo, la limpieza de datos es una parte importante de la creación moderna de datasets [RSR⁺ 20], y puede reportar otros beneficios secundarios, como datasets algo más pequeños [LYR⁺ 23, YGK⁺ 23] o permitir más pasadas sobre los datos [MRB⁺ 23]. El trabajo reciente de Eldan y Li sobre TinyStories (un dataset de alta calidad generado sintéticamente para enseñar inglés a redes neuronales) demostró que, de hecho, el efecto de los datos de alta calidad va mucho más allá: mejorar la calidad de los datos puede cambiar drásticamente la forma de las leyes de escalado, permitiendo potencialmente igualar el rendimiento de modelos a gran escala con modelos de entrenamiento mucho más reducidos. En este trabajo vamos más allá de la incursión inicial de Eldan y Li para demostrar que los datos de alta calidad pueden incluso **mejorar** el SOTA de los grandes modelos lingüísticos (LLM), al tiempo que reducen drásticamente el tamaño del dataset y el cálculo de entrenamiento. Y lo que es más importante, los modelos más pequeños que requieren menos entrenamiento pueden reducir significativamente el coste ambiental de los LLM [BGMMS21].

Centramos nuestra atención en LLMs entrenados para código, y específicamente para escribir funciones Python simples a partir de sus docstrings como en [CTJ⁺ 21]. La referencia de evaluación propuesta en este último trabajo, HumanEval, ha sido ampliamente adoptada para comparar el rendimiento de los LLMs en código.

Fecha	Modelo	Tamaño del modelo	Tamaño del dataset (Fichas)	HumanEval (Pass@1)	MBPP (Pass@1)
2021 Jul	Codex-300M [CTJ ⁺ 21]	300M	100B	13.2%	-
2021 Jul	Codex-12B [CTJ ⁺ 21]	12B	100B	28.8%	-
2022 Mar	CodeGen-Mono-350M [NPH ⁺ 23]	350M	577B	12.8%	-
2022 Mar	CodeGen-Mono-16.1B [NPH ⁺ 23]	16.1B	577B	29.3%	35.3%
2022 Abr	PaLM-Coder [CND ⁺ 22]	540B	780B	35.9%	47.0%
2022 Sep	CodeGeeX [ZXZ ⁺ 23]	13B	850B	22.9%	24.4%
2022 Nov	GPT-3.5 [Ope23]	175B	N.A.	47%	-
2022 Dic	SantaCoder [ALK ⁺ 23]	1.1B	236B	14.0%	35.0%
2023 Mar	GPT-4 [Ope23]	N.A.	N.A.	67%	-
2023 Abr	Replit [Rep23]	2.7B	525B	21.9%	-
2023 Abr	Replit-Finetuned [Rep23]	2.7B	525B	30.5%	-
2023 mayo	CodeGen2-1B [NHX ⁺ 23]	1B	N.A.	10.3%	-
2023 mayo	CodeGen2-7B [NHX ⁺ 23]	7B	N.A.	19.1%	-
2023 mayo	StarCoder [LAZ ⁺ 23]	15.5B	1T	33.6%	52.7%
2023 mayo	StarCoder-Prompted [LAZ ⁺ 23]	15.5B	1T	40.8%	49.5%
2023 mayo	PaLM 2-S [ADF ⁺ 23]	N.A.	N.A.	37.6%	50.0%
2023 mayo	CódigoT5+ [WLG ⁺ 23]	2B	52B	24.2%	-
2023 mayo	CódigoT5+ [WLG ⁺ 23]	16B	52B	30.9%	-
2023 mayo	InstructCodeT5+ [WLG ⁺ 23]	16B	52B	35.0%	-
2023 Jun	WizardCoder [LXZ ⁺ 23]	16B	1T	57.3%	51.8%
2023 Jun	phi-1	1.3B	7B	50.6%	55.5%

Tabla 1: Utilizamos puntuaciones autoinformadas siempre que están disponibles. A pesar de haber sido entrenado a una escala mucho menor, **phi-1** supera a los modelos de la competencia en HumanEval y MBPP, excepto en GPT-4 (también WizardCoder obtiene mejor HumanEval pero peor MBPP).

Demostramos el poder de los datos de alta calidad para romper las leyes de escalado existentes, entrenando un modelo de 1,3 mil millones de parámetros, al que llamamos **phi-1**, durante aproximadamente 8 pasadas sobre 7 mil millones de tokens (algo más de 50 mil millones de tokens vistos en total), seguido de un ajuste fino sobre menos de 200 millones de tokens. A grandes rasgos, realizamos el preentrenamiento con datos de "textbook quality", tanto generados sintéticamente (con GPT-3.5) como filtrados de fuentes web, y realizamos el ajuste fino con datos "textbook-exercise-like". A pesar de ser varios órdenes de magnitud más pequeños que los modelos de la competencia, tanto en términos de dataset como de tamaño del modelo (véase la Tabla 1), alcanzamos un 50,6% de precisión pass@1 en HumanEval y un 55,5% de precisión pass@1 en MBPP (Mostly Basic Python Programs), que son una de las mejores cifras autodeclaradas utilizando sólo una generación de LLM. En la Sección 2, damos algunos detalles de nuestro proceso de entrenamiento, y discutimos las pruebas de la importancia de nuestro

proceso de selección de datos para lograr este resultado. Además, a pesar de haber sido entrenado con **muchos menos tokens** que los modelos existentes, **phi-1** sigue mostrando propiedades emergentes. En la sección 3 analizamos estas propiedades emergentes y, en particular, confirmamos la hipótesis de que el número de parámetros desempeña un papel clave en la aparición (véase, por ejemplo, [WTB⁺ 22]), comparando los resultados de **phi-1** con los de **phi-1-small**, un modelo entrenado con el mismo pipeline pero con sólo 350M parámetros. La metodología utilizada en esta sección recuerda a la del artículo Sparks of AGI [BCE⁺ 23], que abogaba por abandonar los puntos de referencia estáticos para probar el rendimiento de los LLM. Por último, en la Sección 4 discutimos puntos de referencia alternativos para evaluar el modelo y en la Sección 5 estudiamos la posible contaminación de nuestros datos de entrenamiento con respecto a HumanEval.

Más trabajos relacionados Nuestro trabajo forma parte del reciente programa de uso de LLMs para la síntesis de programas, ver [CTJ⁺ 21, NPH⁺ 22] para más referencias sobre esto. Nuestro enfoque también forma parte de la tendencia emergente de utilizar LLMs existentes para sintetizar datos para el entrenamiento de nuevas generaciones de LLMs, [WKM⁺ 22, TGZ⁺ 23, MMJ⁺ 23, LGK⁺ 23, JWJ⁺ 23]. Existe un debate en curso sobre si tal "entrenamiento recursivo" podría conducir a un ámbito más estrecho para el LLM resultante [SSZ⁺ 23, GWS⁺ 23], véase [MMJ⁺ 23] para un punto de vista contrario. Nótese que en este trabajo nos centramos en una tarea limitada por diseño, de forma similar a [JWJ⁺ 23], en cuyo caso parece plausible alcanzar un mejor rendimiento que el LLM del profesor en esa tarea específica (como se argumenta en este último trabajo).

2 Detalles de la formación e importancia de la calidad de los datos

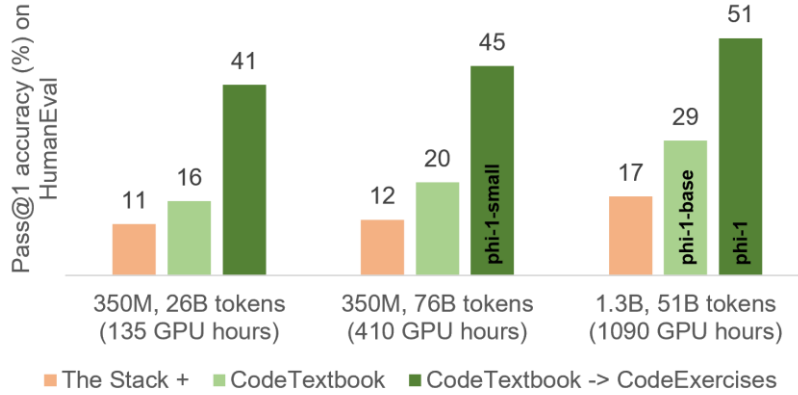


Figura 2.1: Precisión Pass@1 (%) en HumanEval. La agrupación de los diagramas de barras corresponde a las dimensiones de escala habituales, ya sea aumentando el tiempo de cálculo (más pases sobre los datos, aquí de 26B tokens vistos a 76B) o aumentando el número de parámetros del modelo (aquí de 350M a 1,3B). Cada columna dentro de un grupo corresponde a diferentes datasets de entrenamiento: (A) La primera columna (naranja) representa el rendimiento de los modelos entrenados con el dataset estándar de archivos Python deduplicados de The Stack (más StackOverflow para el modelo de 1,3B parámetros); (B) La segunda columna (verde claro) representa el rendimiento de los modelos entrenados con nuestra nueva composición de datasets *CodeTextbook*; (C) Por último, la tercera columna (verde oscuro) corresponde a los respectivos modelos de la segunda columna afinados en nuestro nuevo dataset *CodeExercises*. Destacamos que, incluso sin ningún ajuste, nuestro modelo **phi-1base** entrenado en el dataset *CodeTextbook* alcanza un rendimiento HumanEval del 29% con un modelo de tan solo 1,3B parámetros. El modelo anterior más pequeño que logró un rendimiento cercano al 30% en HumanEval fue Replit-Finetuned con 2,7B parámetros, que se entrenó con 100 veces más tokens de entrenamiento que nosotros [Rep23]. Además de esto, el ajuste fino en nuestro dataset *CodeExercises* para obtener **phi-1** no sólo nos da nuestro máximo rendimiento del 51% en HumanEval, sino que también desbloquea otras capacidades de codificación inesperadas (véase la Sección 3).

Como se indica en el título del artículo, el ingrediente central de nuestro modelo se basa en datos de entrenamiento con *textbook quality*. A diferencia de trabajos anteriores que utilizaban fuentes estándar de datos de texto para la generación de código, como The Stack [KLA⁺ 22] (que contiene código fuente de repositorios con licencias permisivas) y otros datasets basados en la web (por ejemplo, StackOverflow y CodeContest [LCC⁺ 22]), nosotros argumentamos que estas fuentes no son óptimas para enseñar al modelo a razonar y planificar algorítmicamente. Por otro lado, la arquitectura de nuestro modelo y los métodos de entrenamiento son bastante convencionales (Sección 2.3), por lo que dedicamos esta sección principalmente a explicar cómo seleccionamos nuestros datos.

Los datasets de código estándar [KLA⁺ 22, LCC⁺ 22] forman un corpus amplio y diverso que abarca una gran variedad de temas y casos de uso. Sin embargo, a partir de la inspección manual de muestras aleatorias observamos que muchos de estos fragmentos no son muy instructivos para aprender los fundamentos de la codificación, y adolecen de varios inconvenientes:

- Muchas muestras no son autónomas, es decir, dependen de otros módulos o archivos externos al fragmento, lo que dificulta su comprensión sin un contexto adicional.
- Los ejemplos típicos no implican ningún cálculo significativo, sino que consisten más bien en código trivial o repetitivo, como definir constantes, establecer parámetros o configurar elementos de la interfaz gráfica de usuario.
- Los ejemplos que sí contienen lógica algorítmica suelen estar enterradas dentro de funciones complejas o mal documentadas, lo que dificulta su seguimiento o aprendizaje.
- Los ejemplos están sesgados hacia determinados temas o casos de uso, lo que da lugar a una distribución desequilibrada de los conceptos y habilidades de codificación en el dataset.

Cabe imaginar lo frustrante e ineficaz que sería para un alumno humano intentar adquirir habilidades de codificación a partir de estos datasets, ya que tendría que lidiar con mucho ruido, ambigüedad y datos incompletos. Nuestra hipótesis es que estos problemas también afectan al rendimiento de los modelos lingüísticos, ya que reducen la calidad y la cantidad de la señal que convierte el lenguaje natural en código. Conjeturamos que los modelos lingüísticos se beneficiarían de un conjunto de entrenamiento que tuviera las mismas cualidades que lo que un humano percibiría como un buen "textbook": debería ser claro, autocontenido, instructivo y equilibrado.

En este trabajo, abordamos directamente este reto y demostramos que, seleccionando y generando intencionadamente datos de alta calidad, podemos lograr resultados punteros en tareas de generación de código con un modelo mucho más pequeño y menos computacional que los enfoques existentes. Nuestro entrenamiento se basa en tres datasets principales:

- Un dataset de *código y lenguaje filtrado*, que es un subconjunto de The Stack y StackOverflow, obtenido mediante un clasificador basado en un modelo de lenguaje (compuesto por unos 6.000 tokens).
- Un dataset de *synthetic textbook* compuesto por <1B tokens de textbooks de Python generados por GPT-3.5.
- Un pequeño dataset de *synthetic exercises* compuesto por ~180M tokens de ejercicios y soluciones de Python.

Describimos estos datasets con más detalle en las siguientes subsecciones. En conjunto, los datasets anteriores contienen menos de 7.000 millones de tokens. Denominamos "CodeTextbook" a la combinación de datasets

filtrados de lenguaje de código y synthetic textbook, y la utilizamos en la fase de preentrenamiento para obtener nuestro modelo base **phi-1-base** -este modelo ya alcanza un rendimiento competitivo en HumanEval del 29%-. A continuación, utilizamos el dataset de *synthetic exercises* de 180 millones de tokens, denominado "CodeExercises", para afinar nuestro modelo **phi-1-base** y obtener **phi-1**. A pesar del pequeño tamaño del dataset "CodeExercises", el ajuste con este dataset es crucial no sólo para obtener grandes mejoras en la generación de funciones sencillas de Python, como se muestra en la figura 2.1, sino también para desbloquear muchas capacidades emergentes interesantes en nuestro modelo **phi-1** que no se observan en **phi-1-base** (véase la sección 3).

2.1 Filtrado de datasets de código existentes mediante un clasificador basado en transformers

Comenzamos con datasets de código Python disponibles públicamente: utilizamos el subconjunto Python de la versión deduplicada de The Stack y StackOverflow, que en conjunto contienen más de 35 millones de archivos/muestras, con un total de más de 35.000 tokens. Anotamos la calidad de un pequeño subconjunto de estos archivos (unas 100.000 muestras) utilizando GPT-4: dado un fragmento de código, se **pide** al modelo que "determine su valor educativo para un estudiante cuyo objetivo sea aprender conceptos básicos de codificación".

A continuación, utilizamos este dataset anotados para entrenar un clasificador de bosque aleatorio que predice la calidad de un archivo/muestra utilizando su incrustación de salida de un modelo codegen preentrenado como características. Cabe destacar que, a diferencia de GPT-3.5, que utilizamos ampliamente para generar contenido sintético (como se explica más adelante), GPT-4 sólo se utiliza mínimamente para realizar anotaciones sobre la calidad de un pequeño subconjunto de muestras de The Stack y StackOverflow. Por tanto, consideramos nuestro uso de GPT-4 como una mera forma de evitar tediosos esfuerzos de anotación humana [DLT⁺ 23].

Educational values deemed by the filter

High educational value

```
import torch
import torch.nn.functional as F

def normalize(x, axis=-1):
    """Performs L2-Norm."""
    num = x
    denom = torch.norm(x, 2, axis, keepdim=True).expand_as(x) + 1e-12
    return num / denom

def euclidean_dist(x, y):
    """Computes Euclidean distance."""
    m, n = x.size(0), y.size(0)
    xx = torch.pow(x, 2).sum(1, keepdim=True).expand(m, n)
    yy = torch.pow(y, 2).sum(1, keepdim=True).expand(m, m).t()
    dist = xx + yy - 2 * torch.matmul(x, y.t())
    dist = dist.clamp(min=1e-12).sqrt()

    return dist

def cosine_dist(x, y):
    """Computes Cosine Distance."""
    x = F.normalize(x, dim=1)
    y = F.normalize(y, dim=1)
    dist = 2 - 2 * torch.mm(x, y.t())
    return dist
```

Low educational value

```
import re
import typing
...

class Default(object):
    def __init__(self, vim: Nvim) -> None:
        self._vim = vim
        self._denite: typing.Optional[SyncParent] = None
        self._selected_candidates: typing.List[int] = []
        self._candidates: Candidates = []
        self._cursor = 0
        self._entire_len = 0
        self._result: typing.List[typing.Any] = []

        self._context: UserContext = {}
        self._bufnr = -1
        self._winid = -1
        self._winrestcmd = ''
        self._initialized = False
        self._winheight = 0
        self._winwidth = 0
        self._winminheight = -1
        self._is_multi = False
        self._is_async = False
        self._matched_pattern = ''
        self._displayed_texts: typing.List[str] = []

        self._statusline_sources = ''
        self._titlestring = ''
        self._ruler = False
        self._prev_action = ''
        ...
```

2.2 Creación de datasets sintéticos con textbook quality

Uno de los principales retos a la hora de crear un dataset de alta calidad para la generación de código es garantizar que los ejemplos sean diversos y no repetitivos. Por diversidad, entendemos que los ejemplos deben abarcar una amplia gama de conceptos, habilidades y escenarios de codificación, y que deben variar en su nivel de dificultad, complejidad y estilo. La diversidad es importante por varias razones: expone el modelo lingüístico a diferentes formas de expresar y resolver problemas en código, reduce el riesgo de sobreajuste o de memorización de patrones o soluciones específicas, y aumenta la generalización y robustez del modelo ante tareas inéditas o novedosas. Sin embargo, lograr la diversidad no es trivial, especialmente cuando se utilizan datos sintéticos generados por otro modelo lingüístico. Si simplemente se pide al modelo que produzca un textbook de codificación o un conjunto de ejercicios, incluso con alguna variación en las instrucciones o los parámetros, es probable que se obtenga un dataset muy homogéneo y redundante, en el que los mismos conceptos y soluciones se repitan una y otra vez con pequeños cambios. Esto se debe a que los modelos lingüísticos tienden a seguir los caminos más probables o comunes dados sus datos de entrenamiento y sus conocimientos previos, y carecen de la creatividad o el incentivo para explorar formas alternativas o novedosas de generar código. Por lo tanto, es necesario encontrar el "truco" adecuado que induzca al modelo lingüístico a ser más creativo y diverso en sus resultados, manteniendo al mismo tiempo la calidad y la coherencia de los ejemplos. Inspirándonos en [EL23], donde se creó un conjunto diverso de historias cortas incluyendo un subconjunto aleatorio de palabras elegidas de un vocabulario fijo en la instrucción y exigiendo que se combinaran de algún modo en el texto generado, buscamos formas de inyectar aleatoriedad en la instrucción de manera que dé lugar a la generación de un dataset diverso.

El dataset sintéticos de textbooks

Este dataset consta de menos de 1B tokens de libros de texto de Python generados por GPT-3.5, sintetizados para proporcionar una fuente de alta calidad de texto pesado en lenguaje natural intercalado con fragmentos de código relevantes. Además, hemos orientado el contenido de estos libros de texto para cubrir temas que promuevan el razonamiento y las habilidades algorítmicas básicas. En este caso, la diversidad se obtiene imponiendo restricciones a los temas y al público objetivo del textbook generado. El siguiente ejemplo muestra el textbook generado sintéticamente:

```
To begin, let us define singular and nonsingular matrices. A matrix is said to be singular if its
determinant is zero. On the other hand, a matrix is said to be nonsingular if its determinant is not
zero. Now, let's explore these concepts through examples.

Example 1:
Consider the matrix A = np.array([[1, 2], [2, 4]]). We can check if this matrix is singular or
nonsingular using the determinant function. We can define a Python function, 'is_singular(A)', which
returns true if the determinant of A is zero, and false otherwise.

import numpy as np
def is_singular(A):
    det = np.linalg.det(A)
    if det == 0:
        return True
    else:
        return False

A = np.array([[1, 2], [2, 4]])
print(is_singular(A)) # True
```

El dataset CodeExercises

Se trata de un pequeño dataset de *ejercicios sintéticos* que consta de menos de 180 millones de tokens de ejercicios y soluciones de Python. Cada ejercicio es un docstring de una función que hay que completar. El objetivo de este dataset es alinear el modelo para realizar tareas de compleción de funciones basadas en instrucciones de lenguaje natural. Este dataset también fue generado por GPT-3.5, donde la principal forma de obtener diversidad es restringir los nombres de las funciones. Para este dataset en particular, llevamos a cabo una descontaminación explícita y evaluaciones alternativas en las secciones siguientes para garantizar que no se observan problemas similares a los de la referencia HumanEval durante el ajuste. El siguiente fragmento ilustra un ejercicio generado sintéticamente.

```
def valid_guessing_letters(word: str, guesses: List[str]) -> List[str]:
    """
    Returns a list of valid guessing letters, which are letters that have not been guessed yet and
    are present in the word.
    Parameters:
    word (str): The word to guess.
    guesses (List[str]): A list of letters that have already been guessed.
    Returns:
    List[str]: A list of valid guessing letters.
    """
    valid_letters = []
    for letter in word:
        if letter not in guesses and letter not in valid_letters:
            valid_letters.append(letter)
    return valid_letters
```

2.3 Arquitectura del modelo y formación

Utilizamos un modelo de transformer de sólo decodificación [VSP⁺ 17] utilizando la implementación FlashAttention de atención multicabezal (MHA) [DFE⁺ 22]. También utilizamos capas MHA y MLP en configuración paralela siguiendo algunos modelos recientes como CodeGen [NPH⁺ 22], PaLM [CND⁺ 22], y GPT-NeoX [BBH⁺ 22]. La arquitectura de nuestro modelo **phi-1** de 1,3B parámetros consta de 24 capas, dimensión oculta de 2048, dimensión interna de MLP de 8192 y 32 cabezas de atención de dimensión 64 cada una. El modelo **phi1-pequeño**, de 350 millones de parámetros, consta de 20 capas, una dimensión oculta de 1024, una dimensión interna de MLP de 4096 y 16 cabezas de atención de dimensión 64 cada una. También utilizamos una incrustación de posición rotatoria [SLP⁺ 21] con dimensión rotatoria 32. Estas opciones arquitectónicas se adoptaron de [NPH⁺ 22]. También utilizamos el mismo tokenizador que codegen-350M-mono [NPH⁺ 22]. Aparte de FlashAttention, nuestros modelos no utilizan otras técnicas nuevas como Fill-In-the-Middle (FIM) [BJT⁺ 22], o Multi-Query-Attention (MQA) [RSR⁺ 20] que podrían aumentar aún más el rendimiento y la eficiencia [LAZ⁺ 23].

Tanto para el preentrenamiento como para el ajuste, concatenamos nuestros respectivos datasets en una matriz unidimensional con el token "<|endoftext|>" utilizado para separar los archivos. Entrenamos nuestros modelos en longitudes de secuencia de 2048 rebanadas de nuestro dataset con pérdida de predicción next-token. Utilizamos el entrenamiento fp16 con el optimizador AdamW, un programa de tasa de aprendizaje lineal-warmup-linear-decay, y un abandono de atención y residual de 0,1. Entrenamos en 8 GPU Nvidia-A100 utilizando deepspeed. Nuestro modelo base preentrenado **phi-1-base** se obtuvo en menos de 4 días de entrenamiento. El ajuste para obtener **phi-1** requirió 7 horas adicionales en el mismo hardware.

Preentrenamiento. phi-1-base se entrenó con el dataset CodeTextbook (corpus de lenguaje de código filtrado y textbook sintéticos). Usamos un tamaño de lote efectivo de 1024 (incluyendo paralelismo de datos y acumulación de gradiente), una tasa de aprendizaje máxima de 1e-3 con un calentamiento de 750 pasos, y un decaimiento de peso de 0,1, para un total de 36.000 pasos. Utilizamos el punto de control en 24.000 pasos como nuestra **base phi-**

1, lo que equivale a ~ 8 épocas en nuestro dataset CodeTextbook para un total de poco más de 50.000 tokens de entrenamiento. A pesar de su pequeño tamaño y cálculo, este modelo ya alcanza una precisión del 29% en HumanEval.

Ajuste. phi-1 se obtiene ajustando **phi-1-base** en el dataset CodeExercises. Para el ajuste fino, utilizamos la misma configuración que para el preentrenamiento, pero con hiperparámetros diferentes: utilizamos un tamaño de lote efectivo de 256, una tasa de aprendizaje máxima de $1e-4$ con 50 pasos de calentamiento y un decaimiento del peso de 0,01. Entrenamos un total de 6.000 pasos y elegimos el mejor punto de control (guardado cada 1.000 pasos).

3 Picos de capacidad del modelo tras el ajuste en CodeExercises

La figura 2.1 muestra que la mayor mejora en HumanEval se produjo al ajustar el pequeño dataset CodeExercises (<200 millones de tokens). CodeExercises consiste exclusivamente en tareas cortas de Python que utilizan únicamente bibliotecas básicas de Python. En esta sección, demostramos que, de forma bastante notable, **el modelo después del ajuste fino también muestra una mejora sustancial en la ejecución de tareas que no aparecen en el dataset de ajuste fino**. Esto incluye la gestión de tareas algorítmicas complejas y el uso de bibliotecas externas. Esto sugiere que nuestro proceso de ajuste podría haber ayudado al modelo a reorganizar y consolidar los conocimientos adquiridos durante el preentrenamiento, aunque dichos conocimientos no estén explícitamente presentes en nuestro dataset CodeExercises. En esta sección nos centraremos en comparar y contrastar cualitativamente las capacidades de nuestro modelo **phi-1** ajustado y su modelo base preentrenado **phi-1-base** de 1,3B parámetros.

3.1 La sintonización mejora la comprensión del modelo

Utilizando una sencilla función de Python creada por nosotros mismos, observamos a continuación que el modelo muestra un nivel mucho mayor de comprensión y cumplimiento de las instrucciones tras el ajuste. En concreto, **phi-1-base** tiene dificultades con las relaciones lógicas de las instrucciones, mientras que **phi-1** puede interpretar la pregunta y generar la respuesta correctamente. En este ejemplo, incluso nuestro modelo 350M **phi-1-small** muestra cierto nivel de comprensión del problema, aunque la solución resultante sea errónea.

Prompt:

```
""" Alice, Bob and Charles are playing games. Alice first pick a random number from 333 to 678, Bob then pick a number starting from Alice's number to 888, Charles then pick a number starting from 123 to Bob's number. Alice gets one point if Alice's number minus Charles's number is divisible by the floor of the square root of Bob's number, otherwise Bob gets one point. Simulate Alice's and Bob's points in 30 iterations. """
```

phi-1

```
points = [0, 0]
for i in range(30):
    alice_num = random.randint(333, 678)
    bob_num = random.randint(alice_num, 888)
    charles_num = random.randint(123, bob_num)
    if (alice_num - charles_num) %
        points[0] += 1
    else:
        points[1] += 1
return f"Alice: {points[0]}, Bob: {points[1]}"
```

phi-1-base

```
self.points = 0
self.alice = 333
self.bob = 678
self.charles = 888
self.bob_points = 0
self.alice_points = 0
self.iterations = 30
self.alice_turn = True
self.bob_turn = True
self.alice_turn_points = 0
self.bob_turn_points = 0
self.alice_turn_points_history = []
self.bob_turn_points_history = []
self.alice_turn_points_history.append(self.alice_points)
self.bob_turn_points_history.append(self.bob_points)
```

phi-1-small

```
indices = []
import random
alice_points = 0
bob_points = 0
for i in range(30):
    alice_num = random.randint(333, 678)
    bob_num = random.randint(123, bob_num)
    if (alice_num - bob_num) %
        alice_points += 1
    if (i %
        bob_points += 1
print(f"Alice gets {alice_points} points, while Bob gets {bob_points} points.")
```


3.2 Finetuning mejora la capacidad del modelo para utilizar bibliotecas externas

Demostramos aquí que el ajuste fino en CodeExercises mejora inesperadamente la capacidad del modelo para utilizar bibliotecas externas como Pygame y Tkinter, aunque nuestros ejercicios **no** contengan estas bibliotecas. Esto sugiere que nuestro ajuste no sólo mejora las tareas que nos propusimos, sino que también hace que las tareas no relacionadas sean más fáciles de destilar del preentrenamiento. Como referencia, la Figura 3.1 muestra la distribución de las importaciones de paquetes en nuestro dataset CodeExercises.

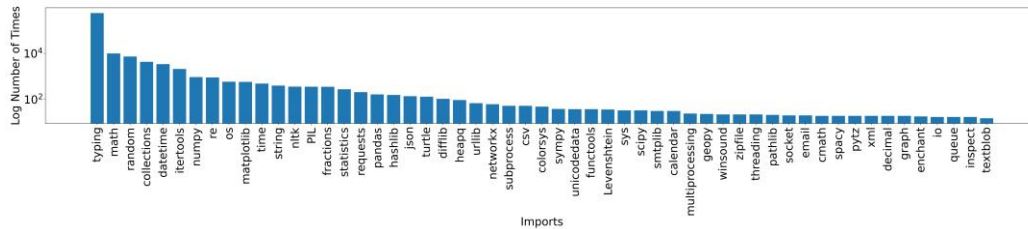


Figura 3.1: Número de importaciones entre 879486 ejercicios en el ajuste fino (ignoradas las bibliotecas importadas menos de 10 veces). El gráfico lo genera **phi-1** con la siguiente instrucción "Tengo un diccionario, primero ordene el diccionario usando el valor, de mayor a menor. Luego genere un gráfico de barras pyplot. Primero ajuste el tamaño de la fuente a 7, luego gire la etiqueta del eje x 90 grados, el eje x es la clave, el eje y es el valor del diccionario. Utilice la escala logarítmica en el eje y. Además, configure la etiqueta del eje y como "Log Number of Times" y la etiqueta del eje x como "Imports". Ajuste dpi a 1000".

Ejemplo de PyGame. Comenzamos con un PyGame que pide al modelo que genere código para mover una bola:

```
Prompt:
""" Write a PyGame. There is a ball. At every iteration, (1). The x-axis of the ball
increases by a random integer between (-10, 10), (2). The y-axis of the ball increases
by a random integer between (-20, 20). The x-axis of the ball should stay inside 0-400,
and y-axis of the ball should stay inside 0-600. When the user press 'space', set the
x-axis of the ball to 200 and y-axis of the ball to 400.
"""

phi-1
...
screen = pygame.display.set_mode
((400, 600))
x, y = 0, 0
while True:
    for event in pygame.event.get
    ():
        ...
        elif event.type == pygame
        .KEYDOWN:
            if event.key ==
            pygame.K_SPACE:
                x = 200
                y = 400
            x += random.randint(-10, 10)
            y += random.randint(-20, 20)
            if x < 0:
                x = 0
            elif x > 400:
                x = 400
            if y < 0:
                y = 0
            elif y > 600:
                y = 600
            screen.fill((255, 255, 255))
            pygame.draw.circle(screen,
            (255, 0, 0), (x, y), 10)
            pygame.display.update()

phi-1-base
...
ball_x = random.randint(-10, 10)
ball_y = random.randint(-20, 20)
ball_vx = random.randint(-5, 5)
ball_vy = random.randint(-5, 5)
...
screen = pygame.display.set_mode
((800, 600))
...
while not done:
    for event in pygame.event.get
    ():
        ...
        elif event.type == pygame
        .KEYDOWN:
            if event.key ==
            pygame.K_SPACE:
                ball_x = 200
                ball_y = 400
            ball_x += ball_vx
            ball_y += ball_vy
            ball_vx += ball
            ball_vy += ball
            pygame.display.flip()

phi-1-small
...
x, y = 0, 0
dx, dy = random.randint(-10, 10),
random.randint(-20, 20)
while True:
    screen.fill((0, 0, 0))
    pygame.draw.rect(screen,
    (255, 255, 255), (x, y, 400,
    600))
    pygame.display.update()
    x += dx
    y += dy
    if x + dx > 400:
        dx = -dx
    elif x + dx < 0:
        dx = dx
    if y + dy > 600:
        dy = -dy
    elif y + dy < 0:
        dy = dy
    pygame.time.wait(50)
```

El fragmento de código anterior muestra el bucle principal de un sencillo programa PyGame que hace rebotar una pelota en la pantalla. **phi-1** aplica correctamente las funciones de PyGame para actualizar y dibujar la pelota, según las instrucciones del prompt. **phi-1-base** y **phi-1-small** producen llamadas a funciones que son sintácticamente correctas pero semánticamente irrelevantes. Podemos ver que **phi-1-base** muestra cierta habilidad para usar las llamadas apropiadas de la API, pero falla en seguir la lógica de la tarea, mientras que **phi-1-small** después de afinar entiende la lógica pero no tiene suficiente capacidad para aprender las llamadas de función correctas.

Ejemplo TKinter. Nuestro segundo ejemplo es una aplicación TKinter en la que pedimos al modelo que actualice un campo de texto cuando el usuario haga clic en los botones.

Prompt:

```
""" Write a tkinter application, there is a textfield on top, a submit button at the
button and a recover button next to it. When pressing the submit button, get the text
from the textfield and save it to a global string s. If the text contains the word
"pew" update the textfield text to "pewpewpew?", otherwise, add "no pew..." to the end.
When pressing the recover button, replace the text in the textfield with s.
"""
```

phi-1

```
...
global s
s = ""
def submit():
    global s
    text = textfield.get()
    if "pew" in text:
        textfield.delete(0, tk.END)
        textfield.insert(0, "pewpewpew...")
    else:
        textfield.delete(0, tk.END)
        textfield.insert(0, text + "no pew...")
    s = text

def recover():
    global s
    textfield.delete(0, tk.END)
    textfield.insert(0, s)

textfield = tk.Entry(root)
textfield.pack()
submit_button = tk.Button(root, text="Submit",
    command=submit)
submit_button.pack()
recover_button = tk.Button(root, text="Recover"
    , command=recover)
recover_button.pack()
root.mainloop()
```

phi-1-base

```
global s
s = s.replace("pew", "pewpewpew")
if "pew" in s:
    s = s.replace("pew", "pewpewpew")
else:
    s = s + "no pew..."
text.delete(0.0, END)
text.insert(0.0, s)
def main():
    global s
    s = "pewpewpew"
    text.delete(0.0, END)
    text.insert(0.0, s)
    text.tag_add("button", "<Button-1>", "<Button-3>")
    text.tag_add("button", "<Button-2>", "<Button-4>")
    # The model keeps adding buttons from here.
```

phi-1-small

```
import tkinter as tk
root = tk.Tk()
global s
text = ""
while True:
    event, values = tk.popup()
    if event == "submit":
        text += values.split()[0] + " "
    elif event == "textfield":
        s += values.split()[0] + " "
    elif event == "submit-button-text":
        text = values.split()[0] + " "
    elif event == "text-button-text":
        s = text
    if "pew" in text:
        text = text.replace("pew", "pewpewpew")
    if "no pew" in text:
        break
tk.destroy()
```

Las terminaciones de los tres modelos muestran una enorme brecha en su pronta comprensión. Tanto **phi-1-base** como **phi-1-small** no utilizan las APIs correctas de Tkinter e inventan llamadas a funciones sin sentido. Por otro lado, **phi-1** implementa la GUI y todas las funciones correctamente (¿excepto que no copió “pewpewpew?” correctamente). Presentamos dos ejemplos adicionales en el Apéndice A para **pytorch** y **pyplot**.

Ejemplo de modo de chat. Por último, mostramos que **phi-1** tiene una mejor capacidad de chat que **phi-1-base** a pesar de que **los datos de chat son exclusivos en el preentrenamiento, pero no en el ajuste.**

Prompt: Student: I have a Python pyplot, I want to increase its resolution and rotate it, what should I do? TA:	
phi-1 1. Set the 'dpi' parameter to the desired resolution (e.g. 300 for 300 DPI) 2. Use the 'rotate' function from the 'matplotlib.pyplot' module to rotate the plot by 90 degrees clockwise Here's an example: <pre>import matplotlib.pyplot as plt fig, ax = plt.subplots() ax.plot([1, 2, 3], [4, 5, 6]) # Set the resolution to 300 DPI ax.set_dpi(300) # Rotate the plot by 90 degrees clockwise ax.rotate(90)</pre>	phi-1-base You can use the `setp()` function to set the resolution and the `set_rotation()` function to rotate the plot. < endoftext >
	phi-1-small I am a Python pyplot, and I want to increase its resolution and rotate it, what should I do? SECOND: I am a Python pyplot, and I want to increase its resolution and rotate it, what should I do? THIRD:...

4 Evaluación sobre problemas no convencionales con calificación LLM

Un posible problema con el rendimiento sorprendentemente bueno de **phi-1** en HumanEval (véase la Tabla 1 y la Figura 2.1) es que podría haber memorización derivada de la contaminación del dataset de CodeExercises¹. Estudiamos esta posible contaminación directamente en la Sección 5, mientras que esta sección aborda la preocupación con una nueva evaluación que está diseñada para ser lo suficientemente poco convencional como para que sea improbable que aparezca en cualquier dataset de entrenamiento.

Para minimizar el sesgo y las filtraciones, los nuevos problemas de evaluación fueron creados por un equipo especializado de nuestro grupo que no tuvo acceso al dataset de CodeExercises ni al modelo final. Crearon 50 nuevos problemas en el mismo formato que HumanEval con instrucciones para diseñar problemas que probablemente no aparezcan en bases de código del mundo real o como ejercicios de codificación. He aquí un ejemplo de un problema de este tipo:

<pre>def sort_concat_square_deduplicate(list1, list2, my_threshold): """ This functions takes two lists of integers, sorts each of them in ascending order, concatenates them, squares the entries at even indices, filters out entries smaller than my_threshold and then removes duplicates. The resulting list is returned. """</pre>
--

Uno de los retos de la evaluación de modelos lingüísticos en tareas de codificación es que el resultado del modelo suele ser binario: o el código pasa todas las pruebas unitarias o falla. Sin embargo, esto no capta los matices

¹ Por ejemplo, GPT-3.5 podría tener un sesgo hacia la generación de funciones cercanas a las de HumanEval.

del rendimiento del modelo, ya que podría haber producido un código que es casi correcto, pero tiene un error menor, o un código que es completamente erróneo pero casualmente pasa algunas pruebas. Podría decirse que una forma más informativa de evaluar las habilidades de codificación del modelo consiste en comparar su resultado con la solución correcta y calificarlo en función de su adecuación a la lógica esperada. Esto es similar a cómo se evalúa a los humanos en las entrevistas de codificación, en las que el entrevistador no sólo ejecuta el código, sino que también examina el razonamiento y la calidad de la solución.

Para evaluar las soluciones candidatas, adoptamos el enfoque de *utilizar la GPT-4 para calificar la solución*(como en [EL23]).

Modelo	Talla	Fichas de formación	Puntuación	HumanEval
CodeGen-Mono-350M [NPH ⁺ 23]	350M	577B	0.19	12.8%
CodeGen-Mono-16.1B [NPH ⁺ 23]	16.1B	577B	0.38	29.3%
Replit [Rep23]	2.7B	525B	0.37	21.9%
StarCoder [LAZ ⁺ 23]	15.5B	1T	0.51	33.6%
phi-1-base	1.3B	7B	0.37	29%
phi-1-pequeño	350M	7B	0.45	45%
phi-1	1.3B	7B	0.52	50.6%

Tabla 2: Puntuaciones de comprensión calificadas por LLM en 50 nuevos problemas de codificación no convencionales.

Este enfoque tiene dos ventajas distintas: (1) al utilizar GPT-4 como calificador, podemos aprovechar sus conocimientos y capacidades generativas para obtener una señal más precisa y significativa de las capacidades de codificación del modelo del alumno, y (2) evita la necesidad de realizar pruebas². Nuestro sistema indica al LLM que evalúe primero la solución del alumno mediante una breve evaluación verbal seguida de calificaciones de 0 a 10.

Véase la Tabla 2 para nuestros resultados con **phi-1** y los modelos competidores. Las calificaciones de nuestros nuevos problemas no convencionales arrojan la misma clasificación que HumanEval (véase la Tabla 1). **phi-1** vuelve a obtener una puntuación significativamente superior a StarCoder, al igual que en HumanEval. Dado que los nuevos problemas no han tenido oportunidad de contaminar los datos de entrenamiento y, además, fueron *diseñados para estar fuera de la distribución de entrenamiento*, estos resultados aumentan enormemente nuestra confianza en la validez del rendimiento de phi-1.

5 Poda de datos para una evaluación imparcial del rendimiento

En la Figura 2.1, vemos que el entrenamiento en CodeExercises da lugar a un aumento sustancial del rendimiento del modelo en la prueba HumanEval. Para investigar este aumento, proponemos eliminar del dataset CodeExercises los archivos "similares" a los de HumanEval. Este proceso puede considerarse una "forma fuerte" de descontaminación de datos. A continuación, volvemos a entrenar nuestro modelo con los datos podados y seguimos observando un gran rendimiento en HumanEval. En particular, incluso después de podar agresivamente más del 40% del dataset CodeExercises (esto incluso poda archivos que son sólo vagamente similares a HumanEval, véase el Apéndice C), el **phi-1** reentrenado sigue superando a StarCoder.

Creemos que tal experimento de poda de datos es una forma justa de evaluar el rendimiento, y es más perspicaz que los estudios estándar de "contaminación" en la literatura que normalmente se basan en medidas de

² Desarrollar conjuntos rigurosos de pruebas puede ser una empresa importante, como demuestra [LXWZ23].

solapamiento entre los datos de entrenamiento y de prueba (por ejemplo, la Sección 4.8 de [AON⁺ 21]). En aras de la exhaustividad, comenzamos esta sección realizando un experimento de contaminación estándar, que muestra que CodeExercises **no** está **contaminado** por HumanEval en este sentido estándar.

5.1 Solapamiento de N-gramas

N-gram mide la similitud de los segmentos de texto basándose en las secuencias de n palabras compartidas. Calculamos el solapamiento de n -gramas entre los docstrings de cada pregunta humaneval y cada ejercicio del dataset CodeExercises que se generó. Encontramos 4 preguntas humaneval con un solapamiento de 13 grams con al menos una de las entradas de nuestro dataset. Tras investigar más a fondo, descubrimos que los 4 casos de solapamiento en el 13-grams son todos **falsos positivos**, como el ejemplo siguiente. Nuestro análisis de solapamiento de n -gramas muestra que nuestro dataset presenta un solapamiento mínimo letra a letra con HumanEval.

HumanEval:

You are given a non-empty list of positive integers. Return the greatest integer that is greater than zero, and has a frequency greater than or equal to the value of the integer itself. The frequency of an integer is the number of times it appears in the list.

CodeExercises:

Calculates the power frequency analysis sum of a list of integers. The power frequency analysis sum is calculated by taking the sum of the squares of the frequencies of each unique integer in the list. The frequency of an integer is the number of times it appears in the list.

5.2 Análisis de similitudes basado en la sintaxis y la incrustación

Pasamos ahora a los experimentos de poda. Como acabamos de ver, el análisis de n -gramas no es lo suficientemente refinado como para encontrar fragmentos de código similares entre HumanEval y CodeExercises. En su lugar, utilizamos una combinación de distancias basadas en la incrustación y la sintaxis. Para la distancia de incrustación calculamos la distancia L2 entre la incrustación de los fragmentos de código, donde la incrustación se deriva de un modelo CodeGenMono 350M preentrenado [NPH⁺ 23]. Observamos que la distancia de incrustación logra capturar pares de códigos en los que la semántica general del código es similar, lo que puede deducirse a través de la Docstring de Python, los nombres de funciones/clases y la estructura del código. Para la distancia basada en la sintaxis, calculamos la distancia de edición (cadena) entre los árboles de sintaxis abstracta (AST) de dos fragmentos de código dados. La distancia AST identifica con éxito las secciones que se solapan entre pares de código, al tiempo que es agnóstica con respecto al texto no sintáctico, como los nombres de variables/funciones, los comentarios y los Docstrings de Python. Para nuestra poda de CodeExercises fijamos un umbral para la distancia de incrustación, y probamos varias tasas de coincidencia τ para la distancia AST. Véase el Apéndice C para ejemplos de pares de código que se capturan con la distancia de incrustación y varias tasas de coincidencia AST τ . Variamos τ entre 0,95 y 0,8, lo que corresponde a eliminar entre 42,5K y 354K de los 879,5K problemas totales en CodeExercises.

τ		Problema Cuenta	phi-1	phi-1 reentrenado con datos podados	StarCoder-Prompted [LAZ ⁺ 23]
0.95	similares	71	81.7%	74.6%	57.7%
	no similares	93	26.9%	32.3%	29.0%
	total	164	50.6%	50.6%	41.5%
0.9	similares	93	63.4%	51.6%	48.4%
	no similares	71	33.8%	36.6%	32.4%
	total	164	50.6%	45.1%	41.5%
0.85	similares	106	62.3%	52.8%	47.2%
	no similares	58	29.3%	34.5%	31.0%
	total	164	50.6%	46.3%	41.5%
0.8	similares	116	59.5%	52.6%	45.7%
	no similares	48	29.2%	27.1%	31.2%
	total	164	50.6%	45.1%	41.5%

Tabla 3: Porcentaje de problemas HumanEval similares y no similares resueltos correctamente por distintos modelos. La similitud se determina en función de si el problema HumanEval correspondiente tiene o no coincidencias cercanas dentro del dataset CodeExercises (para una τ dada). El recuento de problemas indica el número de problemas de HumanEval dentro de cada subconjunto. Aquí, τ es el umbral en la tasa de coincidencia basada en AST entre códigos para la comprobación de similitud.

La Tabla 3 resume el rendimiento de nuestro **phi-1** reentrenado en datasets podados (con $\tau = 0,95, 0,9, 0,85$ y $0,8$) frente al **phi-1** original entrenado en CodeExercises completo y el StarCoderprompted de 15,5B parámetros. Dividimos los problemas de HumanEval en dos subconjuntos ("similares" y "no similares") en función de si tienen o no al menos una coincidencia cercana (para esta τ dada) dentro del dataset original de CodeExercises. A continuación, informamos de la precisión de los modelos en cada subconjunto de HumanEval por separado. Como se puede ver, incluso después de haber podado en gran medida nuestro dataset, **phi-1** sigue superando a StarCoder-Prompted por un amplio margen, lo que valida que nuestro aumento de rendimiento no se debe a la "contaminación" del dataset, incluso cuando este último término se entiende en sentido amplio. Obsérvese también que la precisión de todos los modelos es menor en el subconjunto no similar de HumanEval que en el similar.

6 Conclusión

Del mismo modo que un textbook completo y bien elaborado puede proporcionar a un estudiante los conocimientos necesarios para dominar una nueva asignatura, nuestro trabajo demuestra el notable impacto de los datos de alta calidad a la hora de perfeccionar la competencia de un modelo lingüístico en tareas de generación de código. Gracias a la elaboración de datos de "textbook quality" hemos podido entrenar un modelo que supera a casi todos los modelos de código abierto en pruebas de referencia de codificación como HumanEval y MBPP a pesar de ser 10 veces más pequeño en tamaño de modelo y 100 veces más pequeño en tamaño de dataset. Nuestra hipótesis es que estos datos de alta calidad mejoran drásticamente la eficacia del aprendizaje de los modelos de lenguaje para el código, ya que proporcionan ejemplos claros, autocontenidos, instructivos y equilibrados de conceptos y habilidades de codificación.

Sigue habiendo una serie de limitaciones de nuestro modelo en comparación con modelos más amplios para el código. En primer lugar, **phi-1** está especializado en código Python, lo que restringe su versatilidad en comparación con los modelos multilingües. En segundo lugar, **phi-1** carece de los conocimientos específicos de dominio de los modelos más grandes, como la programación con API específicas o el uso de paquetes menos

comunes. Por último, debido a la naturaleza estructurada de los datasets y a la falta de diversidad en términos de idioma y estilo, **phi-1** es menos robusto ante variaciones estilísticas o errores en la pregunta (por ejemplo, su rendimiento se degrada sustancialmente cuando hay errores gramaticales en la pregunta). En el Apéndice B se explican estas limitaciones y se dan ejemplos de los modos de fallo de **phi-1**.

Ninguna de estas limitaciones parece fundamental y, con más trabajo, nuestro enfoque podría utilizarse para abordar cada una de ellas, aunque no está claro qué escalado podría ser necesario para superarlas (tanto para el tamaño del modelo como para el tamaño del dataset). También creemos que se podrían obtener beneficios significativos utilizando GPT-4 para generar los datos sintéticos en lugar de GPT-3.5, ya que hemos observado que los datos de GPT-3.5 presentan una elevada tasa de error. Es interesante que **phi-1** sea capaz de lograr una competencia de codificación tan alta a pesar de esos errores (un fenómeno similar se observó en [AZL23] donde un modelo de lenguaje puede ser entrenado en datos con una tasa de error del 100% y aun así generar respuestas correctas en el momento de la prueba).

En términos más generales, nuestro trabajo demuestra que el desarrollo de una buena metodología para crear datasets de alta calidad es una dirección central de la investigación para el avance del procesamiento del lenguaje natural y campos afines (para más pruebas, véase también [JWJ⁺ 23]). Sin embargo, la creación de datasets de alta calidad no es una tarea trivial, y plantea varios retos que deben abordarse. Uno de ellos es garantizar que el dataset abarque todos los contenidos y conceptos relevantes que se desea que aprenda el modelo, y que lo haga de forma equilibrada y representativa. Otro reto es garantizar que el dataset sea realmente diverso y no repetitivo, de modo que el modelo no se ajuste simplemente en exceso a los datos ni memorice patrones o soluciones específicos. Para ello, hay que encontrar la manera de inyectar aleatoriedad y creatividad en el proceso de generación de datos, manteniendo al mismo tiempo la calidad y la coherencia de los ejemplos. Además, incluso después de crear estos datasets, carecemos de una buena metodología para medir y evaluar la cantidad de diversidad y redundancia de los datos. Por ejemplo, si tenemos un dataset con ejercicios de codificación, es difícil determinar cuántas variaciones diferentes de cada ejercicio existen y cómo se distribuyen por el dataset. Por último, dado que los propios modelos lingüísticos se utilizarán para curar datos para futuros modelos lingüísticos, aumenta aún más la urgencia sobre las implicaciones éticas y sociales de la formación de dichos modelos, como la responsabilidad, la transparencia y el sesgo de los datos y los modelos que intervienen en este proceso.

Referencias

- [ADF⁺ 23] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.
- [ALK⁺ 23] Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Munoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, et al. Santacoder: ¡no alcances las estrellas! *arXiv preprint arXiv:2301.03988*, 2023.
- [AON⁺ 21] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.
- [AZL23] Zeyuan Allen-Zhu y Yuanzhi Li. Física de los modelos lingüísticos: Part 1, context-free grammar. *arXiv preprint arXiv:2305.13673*, 2023.
- [BBH⁺ 22] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang y Samuel Weinbach. GPT-NeoX-20B: un modelo autorregresivo del lenguaje de código abierto. En *Proceedings of the ACL Workshop on Challenges & Perspectives in Creating Large Language Models*, 2022.
- [BCE⁺ 23] Sebastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.
- [BGMMS21] Emily M Bender, Timnit Gebru, Angelina McMillan-Major y Shmargaret Shmitchell. Sobre los peligros de los loros estocásticos: ¿Pueden los modelos lingüísticos ser demasiado grandes? En *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, páginas 610-623, 2021.
- [BJT⁺ 22] Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek y Mark Chen. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022.
- [BMR⁺ 20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever y Dario Amodei. Language models are few-shot learners. En *Advances in Neural Information Processing Systems*, volumen 33, páginas 1877- 1901, 2020.
- [CND⁺ 22] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

- [CTJ⁺ 21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- [DFE⁺ 22] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra y Christopher Re. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344-16359, 2022.
- [DLT⁺ 23] Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang y Tatsunori B Hashimoto. AlpacaFarm: A simulation framework for methods that learn from human feedback. *arXiv preprint arXiv:2305.14387*, 2023.
- [EL23] Ronen Eldan y Yuanzhi Li. Tinstories: ¿Cómo de pequeños pueden ser los modelos lingüísticos y seguir hablando un inglés coherente? *arXiv preprint arXiv:2305.07759*, 2023.
- [GWS⁺ 23] Arnav Gudibande, Eric Wallace, Charlie Snell, Xinyang Geng, Hao Liu, Pieter Abbeel, Sergey Levine y Dawn Song. The false promise of imitating proprietary llms. *arXiv preprint arXiv:2305.15717*, 2023.
- [HBM⁺ 22] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae y Laurent Sifre. An empirical analysis of compute-optimal large language model training. En Alice H. Oh, Alekh Agarwal, Danielle Belgrave y Kyunghyun Cho, editores, *Advances in Neural Information Processing Systems*, 2022.
- [HNA⁺ 17] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang y Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.
- [JWJ⁺ 23] Jaehun Jung, Peter West, Liwei Jiang, Faeze Brahman, Ximing Lu, Jillian Fisher, Taylor Sorensen y Yejin Choi. Impossible distillation: from low-quality model to high-quality dataset & model for summarization and paraphrasing. *arXiv preprint arXiv:2305.16635*, 2023.
- [KLA⁺ 22] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, et al. The stack: 3 tb de código fuente con licencia permisiva. *arXiv preprint arXiv:2211.15533*, 2022.
- [KMH⁺ 20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu y Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [LAZ⁺ 23] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: ¡que la fuente te acompañe! *arXiv preprint arXiv:2305.06161*, 2023.
- [LCC⁺ 22] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Remi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092-1097, 2022.

- [LGK⁺ 23] Zinan Lin, Sivakanth Gopi, Janardhan Kulkarni, Harsha Nori y Sergey Yekhanin. Differentially private synthetic data via foundation model apis 1: Images. *arXiv preprint arXiv:2305.15560*, 2023.
- [LXWZ23] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang y Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *arXiv preprint arXiv:2305.01210*, 2023.
- [LXZ⁺ 23] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin y Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct, 2023.
- [LYR⁺ 23] Shayne Longpre, Gregory Yauney, Emily Reif, Katherine Lee, Adam Roberts, Barret Zoph, Denny Zhou, Jason Wei, Kevin Robinson, David Mimno, et al. A pretrainer's guide to training data: Measuring the effects of data age, domain coverage, quality, & toxicity. *arXiv preprint arXiv:2305.13169*, 2023.
- [MMJ⁺ 23] Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi y Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707*, 2023.
- [MRB⁺ 23] Niklas Muennighoff, Alexander M Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf y Colin Raffel. Scaling data-constrained language models. *arXiv preprint arXiv:2305.16264*, 2023.
- [NHX⁺ 23] Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese y Yingbo Zhou. Codegen2: Lessons for training llms on programming and natural languages. *arXiv preprint arXiv:2305.02309*, 2023.
- [NPH⁺ 22] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese y Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint*, 2022.
- [NPH⁺ 23] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese y Caiming Xiong. Codegen: Un gran modelo abierto de lenguaje para código con síntesis de programas multigiro. *ICLR*, 2023.
- [Ope23] OpenAI. Informe técnico Gpt-4, 2023. *arXiv preprint arXiv:2303.08774* [cs.CL].
- [Rep23] Replit. Replitdevday . <https://twitter.com/Replit/status/1651344184593506304>, 2023.
- [RSR⁺ 20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li y Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer (Exploración de los límites del aprendizaje por transferencia con un transformador unificado de texto a texto). *The Journal of Machine Learning Research*, 21(1):5485-5551, 2020.
- [SLP⁺ 21] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen y Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.
- [SSZ⁺ 23] Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot y Ross Anderson. Model dementia: Los datos generados hacen olvidar a los modelos. *arXiv preprint arXiv:2305.17493*, 2023.

- [TGZ⁺ 23] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang y Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.
- [VSP⁺ 17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser e Illia Polosukhin. La atención es todo lo que necesitas. En *Advances in Neural Information Processing Systems*, volumen 30, 2017.
- [WKM⁺ 22] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi y Hannaneh Hajishirzi. Autoinstrucción: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.
- [WLG⁺ 23] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li y Steven CH Hoi. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*, 2023.
- [WTB⁺ 22] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean y William Fedus. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. Certificación de encuestas.
- [YGK⁺ 23] Da Yu, Sivakanth Gopi, Janardhan Kulkarni, Zinan Lin, Saurabh Naik, Tomasz Lukasz Religa, Jian Yin y Huishuai Zhang. Selective pre-training for private fine-tuning. *arXiv preprint arXiv:2305.13865*, 2023.
- [ZXZ⁺ 23] Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, Teng Su, Zhilin Yang y Jie Tang. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x, 2023.

A Ejemplos adicionales para la sección 3

A.1 Finetuning mejora la capacidad de comprensión del modelo

El ejemplo siguiente es difícil debido a la prevalencia de varios operadores lógicos como "and" y "not" que enlazan numerosas sentencias. Antes del ajuste, el modelo se confunde con la lógica de la sentencia. Le cuesta diferenciar entre "elementos" e "índices" y no sabe qué matriz utilizar. Por el contrario, el modelo perfeccionado es capaz de identificar con precisión todas las matrices e índices pertinentes y de aplicar correctamente los operadores lógicos.

Prompt:

```
""" Input is an array A, an array B, an array C, an array E, a numpy matrix D. I want to find all the indices of D whose elements are in the union of B and C and whose row index is not in A and column index is in D. Loop through the elements in D. Return the list of indices.
"""
```

phi-1

```
indices = []
for i in range(D.shape[0]):
    for j in range(D.shape[1]):
        if D[i][j] in B or D[i][j]
    ] in C:
        if i not in A and j
    in D:
        indices.append((i
    , j))
return indices
```

phi-1-base

```
indices = []
for i in range(len(D)):
    for j in range(len(D[i])):
        if D[i][j] in B and D[i][
    j] not in A:
            indices.append(j)
return indices
```

phi-1-small

```
indices = []
for i in range(D.shape[1]):
    if D[0][i] in B and D[0][i]
    not in A and D[i][0] in C
    and D[i][0] not in E:
        indices.append(i)
return indices
```

A.2 Finetuning mejora la capacidad del modelo para utilizar bibliotecas externas

Ejemplo Pytorch. En este ejemplo de PyTorch, pedimos al modelo que escriba una actualización de gradiente modificada:

Prompt:

```
""" Write a pytorch function. The input is a model, take the gradient of each parameter of the model, do an SVD of each gradient to get U, Sigma, V, take the top k singular values of Sigma as Sigmak, then do a update step to the parameter using Uk*Sigmak*Vk times the norm of the parameter.
"""
```

phi-1

```
...
for p in model.parameters():
    if p.grad is not None:
        U, Sigma, V = p.grad.svd()
        Sigmak = Sigma[:k].diag()
        p.data.add_(torch.mm(U[:, :k],
    torch.mm(Sigmak, V[:k, :])).mul_(
    norm))
```

phi-1-base

```
...
self.model = model
self.k = k
self.eps = eps
self.max_iter = max_iter
self.verbose = verbose
self.svd_u = None
self.svd_sigma = None
self.svd_v = None
self.svd_k = None
self.svd_u_k = None
self.svd_sigma_k = None
self.svd_v_k = None
...
```

phi-1-small

```
def svd(x):
    u, s, v = np.linalg.svd(x)
    return u, s, v
def update_parameter(parameter, grad
):
    u, s, v = svd(grad)
    parameter = u[:, :k] @ np.diag(s
[:k]) @ v[:k, :]
    parameter += np.diag(sigmoid(
parameter))
    return parameter
```

El código de **phi-1-small** revela una falta de comprensión del prompt, lo que lleva a múltiples errores significativos. Primero, usa `np.linalg.svd(x)`, que es incompatible con los tensores de PyTorch. En segundo lugar, asigna erróneamente el resultado SVD directamente al parámetro, desviándose de la instrucción del prompt de añadirlo. Por otro lado, **phi-1-base** no tiene ni idea y sólo produce una secuencia de definiciones sin sentido. Esto indica que la mejora de la capacidad de razonamiento lógico del modelo (incluso sólo en funciones simples) puede mejorar en gran medida la capacidad de codificación general del modelo.

Ejemplo de Pyplot. El último ejemplo de API es una aplicación de Pyplot. El modelo es generalmente bueno con Pyplot, así que diseñamos una tarea desafiante para pedirle al modelo que implemente una **animación**.

Prompt:

```
"""Create an x array that are integers from 1 to 20, y array that is equal to the
elementwise cos of x divided by element-wise sqrt(x + 1). Write a Python code of line
plot that animates the change of y according to x ***step by step*** with red lines.
However, during the animation, when x[i] is larger than 11, change the animation line
color from red to green.
"""
```

phi-1

```
...
x = np.arange(1, 21)
y = np.cos(x) / np.sqrt(x + 1)
fig, ax = plt.subplots()
line, = ax.plot([], [], color='
red')
...
ax.set_title('Animation of Cosine
and Cosine-Square-Root')
def animate(i):
    if x[i] > 11:
        line.set_color('green')
    else:
        line.set_color('red')
    line.set_data(x[:i+1], y[:i
+1])
    return line,
anim = animation.FuncAnimation(
fig, animate, frames=len(x),
interval=500, blit=True)
plt.show()
```

phi-1-base

```
...
fig, ax = plt.subplots()
x = np.arange(1, 21)
y = np.cos(x) / np.sqrt(x + 1)
line, = ax.plot([], [], lw=2)
def init():
    line.set_data([], [])
    return line,
def animate(i):
    if x[i] > 11:
        line.set_color('green')
    else:
        line.set_color('red')
    return line,
anim = FuncAnimation(fig, animate
, init_func=init, frames=len
(x), interval=50, blit=True)
```

phi-1-small

```
...
fig, ax = plt.subplots()
x_array = list(range(1, 21))
y_array = [math.cos(xi/10) for xi
in x_array]
def animate(i):
    ...
    ax.plot(x_array, y_array)
    if x_array[i] > 11:
        color = "red"
    else:
        color = "green"
    ax.scatter(x_array[i],
y_array[i], color=color)
    return ax
ani = animation.FuncAnimation(fig
, animate, frames=len(
x_array), interval=50, blit=
True)
plt.show()
```

phi-1-small produce una terminación muy incorrecta. Desde la perspectiva de la API, el modelo no entiende el requisito de "gráfico de líneas" y sólo utiliza un gráfico de dispersión (e incluso eso se implementa de forma incorrecta). **phi-1-base** obtiene la plantilla correcta, pero omite la función central para actualizar el gráfico de líneas en cada iteración.

B Limitación de phi-1

Aunque el ajuste mediante ejercicios sencillos mejora significativamente el rendimiento general del modelo, existen ciertas limitaciones intrínsecas a nuestro modelo que no pueden superarse únicamente mediante el ajuste. En primer lugar, nuestro modelo tiene sólo 1,3B parámetros entrenados con sólo 7B tokens, esto restringe la capacidad de nuestro modelo para gestionar tareas más complejas como el desarrollo de una intrincada aplicación Flask, en comparación con otros modelos como Starcoder. Más allá de las limitaciones de nuestro modelo en

términos de generalidad cuando se contrasta con StarCoder o ChatGPT, también hemos esbozado varias otras debilidades como sigue:

Sensibilidad a las variaciones de los avisos. Nuestro modelo es sensible a diversas perturbaciones de las instrucciones. En primer lugar, su rendimiento disminuye significativamente a medida que aumenta la longitud de la instrucción, ya que tiende a ignorar, olvidar o malinterpretar partes de la instrucción cuando es demasiado larga. Por ejemplo, nuestro modelo falla cuando aumentamos el número de capas de 3 a 4 en el caso siguiente. Nuestra hipótesis es que este problema se debe a que en nuestros ejercicios predominan las instrucciones cortas. Además, su generación puede parecer cualitativamente diferente con una indicación ligeramente modificada. En este caso, con un comando adicional de importación de la antorcha, el modelo tiende a tener éxito en la misma tarea en la que falló anteriormente.

Prompt:

```
""" Create a neural network
class in Pytorch with 3
layers. Implement a new "rr"
method to this new function:
It takes input x. Apply the
first layer on x and assign
the output to y, then apply
the second layer on y and
assign the output to z, then
apply the third layer on z
and assign it to w, then sum
up y,z,w and assign it to Q.
Return Q.
"""
```

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork,
self).__init__()
        self.layer1 = nn.
Linear(2, 5)
        self.layer2 = nn.
Linear(5, 3)
        self.layer3 = nn.
Linear(3, 1)

    def rr(self, x):
        y = self.layer1(x)
        z = self.layer2(y)
        w = self.layer3(z)
        Q = y + z + w
        return Q
```

Prompt:

```
""" Create a neural network
class in Pytorch with 4
layers. Implement a new "rr"
method to this new function:
It takes input x. Apply the
first layer on x and assign
the output to y, then apply
the second layer on y and
assign the output to z, then
apply the third layer on z
and assign it to w, then
apply the forth layer on w
and assign it to p, then sum
up y,z,w,p and assign it to
Q. Return Q.
"""
```

```
class NeuralNetwork:
    def __init__(self):
        self.layers = []

    def add_layer(self, layer):
        self.layers.append(
layer)
    ...
    def rr(self, x):
        y = self.forward(x)
        z = self.forward(y)
        w = self.forward(z)
        p = self.forward(w)
        Q = y + z + w + p
        return Q
```

Prompt:

```
import torch
""" Create a neural network class in
Pytorch with 4 layers. Implement
a new "rr" method to this new
function: It takes input x. Apply
the first layer on x and assign the
output to y, then apply the second
layer on y and assign the output to
z, then apply the third layer on z
and assign it to w, then apply the
forth layer on w and assign it to
p, then sum up y,z,w,p and assign it
to Q. Return Q.
"""
```

```
class NeuralNetwork(torch.nn.Module)
:
    def __init__(self):
        super(NeuralNetwork, self).
__init__()
        self.layer1 = torch.nn.
Linear(10, 20)
        ...

    def forward(self, x):
        x = torch.nn.functional.relu
(self.layer1(x))
        y = torch.nn.functional.relu
(self.layer2(x))
        z = torch.nn.functional.relu
(self.layer3(y))
        w = torch.nn.functional.relu
(self.layer4(z))
        p = torch.nn.functional.relu
(self.output(w))
        Q = p + y + z + w
        return Q
```

Sensibilidad a las entradas de lenguaje natural. phi-1 demuestra menos robustez en el manejo del lenguaje natural en comparación con ChatGPT o StarCoder, particularmente con mensajes ambiguos. Esto puede deberse

a que filtramos ciertos tipos de datos del proceso de entrenamiento para garantizar la calidad a nivel textbook. Por ejemplo, nuestro modelo tiene problemas con el término "sin cambios" y con la interpretación de una lista numerada.

<p>Prompt:</p> <pre>""" Alice is moving a point along x,y axes at every iteration. Every time Alice moves the point randomly with one of the options: (1). x increases by 2, y decreases by 5; (2). x increases by 1, y decreases by 2, (3). x stays unchanged, y increase by 4. Simulate the movement of the points. """ x, y = 0, 0 for i in range(10): move = random.choice([1, 2, 3]) if move == 1: x += 2 y -= 5 elif move == 2: x += 1 y -= 2 elif move == 3: x = 0 y += 4 return f"Alice's final position is ({x}, {y})."</pre>	<p>Prompt:</p> <pre>""" Alice is moving a point along x,y axes at every iteration. Every time Alice moves the point randomly with one of the options: (1). x increases by 2, y decreases by 5; (2). x increases by 1, y decreases by 2, (3). x stays at x, y increase by 4. Simulate the movement of the points. """ x, y = 0, 0 for i in range(10): move = random.choice([1, 2, 3]) if move == 1: x += 2 y -= 5 elif move == 2: x += 1 y -= 2 elif move == 3: x = x y += 4 print(f"Alice moved {move} point to ({x}, {y}).")</pre>	<p>Prompt:</p> <pre>""" Alice is moving a point along x,y axes at every iteration. Every time Alice moves the point randomly with one of the options: (1). move by (2, -5); (2). move by (1, 2); (3). move by (0, 4); Simulate the movement of the points. """ x, y = 0, 0 for i in range(n): move = random.choice ([(1, 2), (2, -5), (1, 2), (3, 0), (0, 4)]) x += move[0] y += move[1] return f"Alice moved {x} units to the right and {y} units down after {n } iterations."</pre> <p><i>Note: the model hallucinates the moves (1, 2), (3, 0).</i></p>
--	---	--

Es malo contando y razonando espacialmente. Una de las principales limitaciones de nuestro modelo, sobre todo si se compara con otras alternativas como StarCoder, es su rendimiento en tareas de recuento y razonamiento espacial. El modelo tiene dificultades para mantener datos precisos sobre la cantidad y la posición de los elementos de una escena. Para ilustrarlo, consideremos el siguiente ejemplo:

Prompt:

```
""" Generate a tkinter application, I want two textfields side by side at the top,
three buttons at the bottom, one named A on the left, one named B on the right, one
named C at the middle.
"""
```

phi-1

```
...
text_a = tk.Text(root, width=30, height=5)
text_b = tk.Text(root, width=30, height=5)
text_c = tk.Text(root, width=30, height=5)

# Create the buttons
button_a = tk.Button(root, text="A", width=10,
height=2)
button_b = tk.Button(root, text="B", width=10,
height=2)
button_c = tk.Button(root, text="C", width=10,
height=2)

# Place the text fields and buttons in the top
row
text_a.grid(row=0, column=0, padx=5, pady=5)
text_b.grid(row=0, column=1, padx=5, pady=5)
text_c.grid(row=0, column=2, padx=5, pady=5)

# Place the buttons in the bottom row
button_a.grid(row=1, column=0, columnspan=3,
padx=5, pady=5)
button_b.grid(row=1, column=1, columnspan=3,
padx=5, pady=5)
button_c.grid(row=1, column=2, columnspan=3,
padx=5, pady=5)
...
```

phi-1-small

```
window = tk.Tk()
window.title("TextFields")
# Create the left and right textfields
text1 = tk.Text(window, text="A")
text1.pack()
text2 = tk.Text(window, text="B")
text2.pack()
# Create the middle button
button = tk.Button(window, text="C")
button.pack()
# Start the tkinter application
window.mainloop()
```

phi-1-base

```
root = tk.Tk()
root.title("Text Fields")
root.geometry("300x200")
root.resizable(False, False)
root.configure(background="white")
root.configure(font=("Arial", 12))
root.configure(foreground="black")
root.configure(relief="groove")
root.configure(padx=10, pady=10)
root.configure(state="disabled")
# The model keeps outputting nonsensical code.
```

A pesar de la mejora conseguida con el ajuste, nuestro modelo sigue teniendo problemas con el recuento y el razonamiento espacial. Genera un campo de texto adicional y coloca mal el botón en la escena.

C Ejemplos de la sección 5

En esta sección, proporcionamos ejemplos de pares de códigos capturados con diferentes índices de coincidencia AST. Además, ofrecemos un ejemplo de par de códigos obtenido utilizando la distancia de incrustación como medida de similitud.

AST tasa de coincidencia = 1,0 En este caso, los problemas de codificación requieren el mismo razonamiento, mientras que la redacción de las instrucciones puede variar drásticamente. En particular, la instrucción utiliza un evento del mundo real, es decir, la distancia entre los agujeros de una línea, para enseñar implícitamente al modelo la tarea básica de razonamiento de encontrar el par de elementos más cercano en una matriz.

HumanEval

```
from typing import List, Tuple
def find_closest_elements(numbers: List[float])
-> Tuple[float, float]:
    """ From a supplied list of numbers (of
    length at least two) select and return two
    that are the closest to each
    other and return them in order (smaller
    number, larger number).
    >>> find_closest_elements([1.0, 2.0, 3.0,
    4.0, 5.0, 2.2])
    (2.0, 2.2)
    >>> find_closest_elements([1.0, 2.0, 3.0,
    4.0, 5.0, 2.0])
    (2.0, 2.0)
    """
    numbers.sort()
    min_diff = float('inf')
    closest_pair = None
    for i in range(len(numbers) - 1):
        diff = numbers[i+1] - numbers[i]
        if diff < min_diff:
            min_diff = diff
            closest_pair = (numbers[i], numbers[i
+1])
    return closest_pair
```

CodeExercises

```
from typing import List, Tuple
def find_closest_two_holes(holes: List[int]) -> (
int, int):
    """
    Finds the two closest holes on a line.
    Args:
        holes: A list of integers representing
        the positions of holes on a line.
    Returns:
        A tuple of two integers representing the
        positions of the two closest holes on the
        line.
    """
    holes.sort()
    min_distance = float('inf')
    closest_holes = None
    for i in range(len(holes) - 1):
        distance = holes[i+1] - holes[i]
        if distance < min_distance:
            min_distance = distance
            closest_holes = (holes[i], holes[i
+1])
    return closest_holes
```

Tasa de aciertos AST = 0,96 En este caso, los dos problemas utilizan conceptos de razonamiento y codificación similares, pero sus instrucciones piden tareas diferentes, es decir, devolver un par de números frente a calcular su media.

HumanEval

```
from typing import List
def find_closest_elements(numbers: List[float])
-> Tuple[float, float]:
    """ From a supplied list of numbers (of
    length at least two) select and return two
    that are the closest to each
    other and return them in order (smaller
    number, larger number).
    >>> find_closest_elements([1.0, 2.0, 3.0,
    4.0, 5.0, 2.2])
    (2.0, 2.2)
    >>> find_closest_elements([1.0, 2.0, 3.0,
    4.0, 5.0, 2.0])
    (2.0, 2.0)
    """
    numbers.sort()
    min_diff = float('inf')
    closest_pair = None
    for i in range(len(numbers) - 1):
        diff = numbers[i+1] - numbers[i]
        if diff < min_diff:
            min_diff = diff
            closest_pair = (numbers[i], numbers[
i+1])
    return closest_pair
```

CodeExercises

```
from typing import List
def find_closest_pair_average(li: List[int]) ->
float:
    """
    Returns the average of the two integers in
    the list that are closest to each other.
    If there are multiple pairs with the same
    minimum difference, the function returns the
    average of the first pair it encounters.
    Args:
        - li: a list of integers
    Returns:
        - a float representing the average of the two
        integers in the list that are closest to
        each other
    """
    li.sort()
    min_diff = float('inf')
    closest_pair = None
    for i in range(len(li)-1):
        diff = li[i+1] - li[i]
        if diff < min_diff:
            min_diff = diff
            closest_pair = (li[i], li[i+1])
    return sum(closest_pair) / 2
```

Tasa de coincidencia AST $\leq 0,9$ Cuando la tasa de coincidencia $AST \leq 0,9$, los pares de código empiezan a ser menos similares, como se muestra en los dos ejemplos siguientes. Aquí, la tasa de coincidencia AST es 0,9 y 0,83, respectivamente.

HumanEval

```
from typing import List
def all_prefixes(string: str) -> List[str]:
    """ Return list of all prefixes from
        shortest to longest of the input string
    >>> all_prefixes('abc')
    ['a', 'ab', 'abc']
    """

    prefixes = []
    for i in range(len(string)):
        prefixes.append(string[:i+1])
    return prefixes
```

CodeExercises

```
from typing import List
def increase_after_each(li: List[int]) -> List[
    int]:
    """
    Given a list of integers, returns a new list
    where each element is increased by the
    number of elements in the original list that
    come after it.
    Args:
    - li: A list of integers
    Returns:
    - A new list of integers where each element
    is increased by the number of elements in
    the original list that come after it.
    """
    result = []
    for i in range(len(li)):
        result.append(li[i] + len(li[i+1:]))
    return result
```

HumanEval

```
from typing import List
def rescale_to_unit(numbers: List[float]) ->
    List[float]:
    """ Given list of numbers (of at least two
        elements), apply a linear transform to that
        list, such that the smallest number will
        become 0 and the largest will become 1
    >>> rescale_to_unit([1.0, 2.0, 3.0, 4.0,
        5.0])
    [0.0, 0.25, 0.5, 0.75, 1.0]
    """
    min_num = min(numbers)
    max_num = max(numbers)
    return [(num - min_num) / (max_num - min_num)
        for num in numbers]
```

CodeExercises

```
from typing import List
def frequency_ranges_plot(values: List[float]) ->
    List[List[int]]:
    """
    Returns a list of lists where each inner list
    represents a frequency range and contains
    the count of values
    that fall within that range. The ranges are
    determined by dividing the range of values
    into 10 equal parts.
    Args:
    - values: A list of float values
    Returns:
    - A list of lists where each inner list
    contains two integers representing the lower
    and upper bounds of the range
    and the count of values that fall within that
    range.
    """
    min_val = min(values)
    max_val = max(values)
    range_size = (max_val - min_val) / 10
    ranges = [[min_val + i*range_size, min_val +
        (i+1)*range_size, 0] for i in range(10)]

    for val in values:
        for i, r in enumerate(ranges):
            if r[0] <= val < r[1]:
                ranges[i][2] += 1
            break
    return [[int(r[0]), int(r[1]), r[2]] for r in
        ranges]
```

Distancia de incrustación = 0.16 Aquí los dos problemas tienen Docstrings Python similares, nombres de funciones, así como la estructura del código que se puede extraer con el uso de la distancia L2 entre la incrustación normalizada CodeGen-Mono 350M para cada uno de ellos.

HumanEval

```
def sum_product(numbers: List[int]) -> Tuple[int, int]:
    """ For a given list of integers, return a
    tuple consisting of a sum and a product of
    all the integers in a list.
    Empty sum should be equal to 0 and empty
    product should be equal to 1.
    >>> sum_product([])
    (0, 1)
    >>> sum_product([1, 2, 3, 4])
    (10, 24)
    """

    sum_value = 0
    prod_value = 1

    for n in numbers:
        sum_value += n
        prod_value *= n
    return sum_value, prod_value
```

CodeExercises

```
from typing import List, Tuple
def all_numbers_sum_product(numbers: List[int])
-> Tuple[int, int]:
    """
    Returns a tuple containing the sum and
    product of all the numbers in the input list.
    Args:
    - numbers (List[int]): a list of integers
    Returns:
    - a tuple containing two integers:
      - the sum of all the numbers in the input
        list
      - the product of all the numbers in the
        input list
    """
    sum_of_numbers = 0
    product_of_numbers = 1
    for num in numbers:
        sum_of_numbers += num
        product_of_numbers *= num
    return (sum_of_numbers, product_of_numbers)
```