Resumen

Les presentamos phi-1, un nuevo modelo de lenguaje grande para código, con un tamaño significativamente más pequeño que los modelos de la competencia: phi-1 es un modelo basado en transformadores con 1.3B de parámetros, entrenados durante 4 días en 8 A100's, utilizando una selección de datos de "calidad de libro de texto" de la web (6B de tokens) y libros de texto y ejercicios generados sintéticamente con GPT-3.5 (1B de tokens). A pesar de esta pequeña escala, phi-1logra una precisión pass@1 de 50.6% en HumanEval y 55.5% en MBPP. También muestra propiedades emergentes sorprendentes en comparación con phi-1-base, nuestro modelo antes de nuestra etapa de ajuste en un conjunto de datos de ejercicios de codificación, y phi-1-small, un modelo más pequeño con 350M parámetros entrenados con la misma canalización que phi-1 aun alcanza el 45% en HumanEval.

1 Introducción

El arte de entrenar grandes redes neuronales artificiales ha logrado avances extraordinarios en la última década, especialmente después del descubrimiento de la arquitectura Transformer [VSP+17], sin embargo, la ciencia detrás de este éxito sigue siendo limitada. En medio de una amplia y confusa variedad de resultados, surgió una apariencia de orden casi al mismo tiempo que se introdujeron los Transformers, es decir, que el rendimiento mejora de manera algo predecible a medida que se aumenta la cantidad de cómputo o el tamaño de la red [HNA+17], un fenómeno que ahora se conoce como *leyes de escala* [KMH+20]. La exploración posterior de la escala en el aprendizaje profundo estuvo guiada por estas *leyes de escala* [BMR+20], y los descubrimientos de variantes de estas leyes condujeron a un rápido aumento en el rendimiento [HBM+22]. En este trabajo, siguiendo los pasos de Eldan y Li [EL23], exploramos la mejora que se puede obtener a lo largo de un eje diferente: la *calidad* de los datos. Se sabe desde hace mucho tiempo que los datos de mayor calidad conducen a mejores resultados; por ejemplo, la limpieza de datos es una parte importante de la creación moderna de conjuntos de datos [RSR+20], y puede generar otros beneficios secundarios, como conjuntos de datos algo más pequeños [LYR+23, YGK+23] o permitir más pases de datos [MRB+23]. El trabajo reciente de Eldan y Li en TinyStories (un conjunto de datos de alta calidad generado sintéticamente para enseñar inglés a redes neuronales) demostró que, de hecho, el efecto de los datos de alta calidad se extiende mucho más allá de esto: mejorar la calidad de los datos puede cambiar drásticamente la forma de las leyes de escala, lo que potencialmente permite igualar el rendimiento de modelos a gran escala con modelos/entrenamientos mucho más eficientes. En este trabajo vamos más allá de la incursión inicial de Eldan y Li para mostrar que los datos de alta calidad pueden incluso mejorar el SOTA de los modelos de lenguajes grandes (LLM), al tiempo que reduce drásticamente el tamaño del conjunto de datos y el cálculo de entrenamiento. Es importante destacar que los modelos más pequeños que requieren menos capacitación pueden reducir significativamente el costo ambiental de los LLM [BGMMS21].

Centramos nuestra atención en los LLM entrenados para código y, específicamente, en escribir funciones simples de Python a partir de sus cadenas de documentación como en [CTJ+21]. El punto de referencia de evaluación propuesto en este último trabajo, HumanEval, ha sido ampliamente adoptado para comparar el desempeño de los LLM en código. Demostramos el poder de la alta calidad de datos para rompiendo las leyes de escala existentes entrenando un modelo de 1.3B-parametros, al cual llamamos phi-1 para aproximadamente

Fecha	Modelo	Tamaño del Modelo	Tamaño del Dataset (Tokens)	HumanEval (Pass@1)	MBPP (PassA1)
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-Cod [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	Santa Coder [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	21.9%	-
2023 May	CodeGen2-1B [NHX+ 23]	1B	N.A.	10.3%	-
2023 May	CodeGen2-7B [NHX+ 23]	7B	N.A.	19.1%	
2023 May	StarCoder [LAZ+ 23]	15.5B	1T	33.6%	52.7%
2023 May	StarCoder-Prompted [LAZ+ 23]	15.5B	1T	40.8%	49.5%
2023 May	PaLM 2-S [ADF+ 23]	N.A.	N.A.	37.6%	50.0%
2023 May	CodeT5+ [WLG+ 23]	2B	52B	24.2%	-
2023 May	CodeT5+ [WLG+ 23]	16B	52B	30.9%	-
2023 May	InstructCodeT5+ [WLG+ 23]	16B	52B	35.0%	-
2023 May	WizardCoder [LXZ+ 23]	16B	1T	57.3%	51.8%
2023 May	phi-1	1.3B	7B	50.6%	55.5%

Tabla1: Utilizamos scores self-reported siempre que estén 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 por GPT-4 (también WizarCoder obtiene mejor HumanEval pero peor MBPP).

aproximadamente 8 pases sobre 7B de tokens (un poco más de 50B de tokens en total vistos) seguidos de un ajuste fino en menos de 200B de tokens. En términos generales, entrenamos previamente con datos de “calidad de libro de texto”, tanto generados sintéticamente (con GPT-3.5) como filtrados de fuentes web, y ajustamos en datos similares a un “ejercicio de libro de texto”. A pesar de ser varias órdenes de magnitud más pequeñas que los modelos de la competencia, tanto en términos de conjunto de datos como de tamaño del modelo (ver Tabla 1), alcanzamos 50.6 % la precisión de pass@1 en HumanEval y precisión 55.5% pass@1 en MBPP (Principalmente Programas Básicos en Python), los cuales son uno de los mejores valores auto-informados utilizando solo una generación de LLM. En la Sección 2, brindamos algunos detalles de nuestro proceso de entrenamiento y analizamos la evidencia de la importancia de nuestro proceso de selección de datos para lograr este resultado. Además, a pesar de estar entrenado en muchos menos tokens en comparación con los modelos existentes, phi-1 todavía muestra propiedades emergentes. En la Sección 3 discutimos estas propiedades emergentes y, en particular, confirmamos la hipótesis de que el número de parámetros juega un papel clave en el surgimiento (ver ejemplo [WTB+22]), comparando los resultados phi-1 con los phi-1-small, un modelo

entrenado con la misma canalización, pero con solo 350M parámetros. La metodología utilizada en esta sección recuerda al artículo Sparks of AGI [BCE+23] que abogaba por alejarse de los puntos de referencia estáticos para probar el desempeño de los LLM. Finalmente, 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.

Mas trabajos relacionados Nuestro trabajo es parte del programa reciente de uso de LLM para la síntesis de programas, ver [CTJ+21, NPH+22] para más referencias sobre esto. Nuestro enfoque también es parte de la tendencia emergente de utilizar LLM existentes para sintetizar datos para la formación de nuevas generaciones de LLM, [WKM+22, TGZ+23, MMJ+23, LGK+23, JWJ+23]. Existe un debate en curso sobre si dicha "formación recursiva" podría conducir a un alcance más limitado para el LLM resultante [SSZ+23, GWS+23], ver [MMJ+23] para un punto de vista contrario. Tenga en cuenta que en este artículo nos centramos en una tarea limitada por diseño, de manera similar a [JWJ+23], en cuyo caso parece plausible lograr un mejor desempeño que el profesor LLM en esa tarea específica (como se argumenta en este último artículo).

2 Detalles del entrenamiento y la importancia de datos de alta calidad

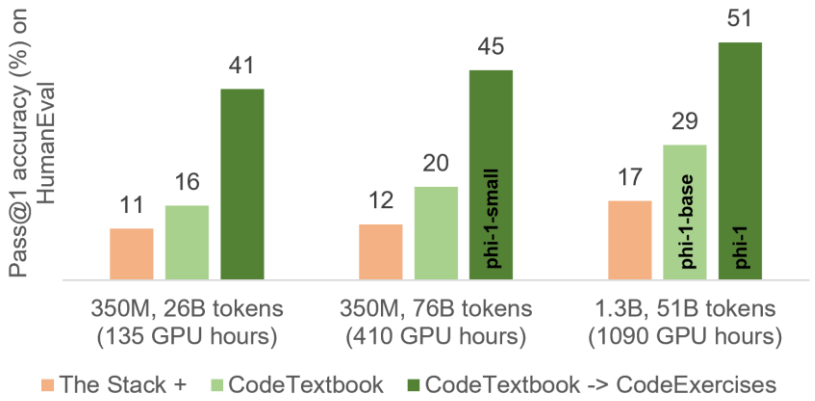

Figura 2.1: Precisión de Pass@1 (%) en HumanEval. La agrupación de gráficos de barras corresponde a las dimensiones de escala habituales de aumentar el tiempo de cálculo (más pases de datos, aquí de 26B de tokens a 76B) o aumentar el número de parámetros del modelo (aquí de 350M a 1.3B). Cada columna dentro de un grupo corresponde a diferentes conjuntos de datos de entrenamiento: (A) La primera columna (naranja) representa el rendimiento de los modelos entrenados en el conjunto de datos estándar de archivos Python des duplicados de The Stack (más StackOverflow para 1.3B parámetros del modelo); (B) La segunda columna (verde claro) representa el rendimiento de los modelos entrenados con nuestra nueva composición del conjunto de datos *CodeTextbook* ; (C) Finalmente, la tercera columna (verde oscuro) corresponde a los respectivos modelos de la segunda columna ajustados en nuestro nuevo conjunto de datos CodeExercises. Destacamos que incluso sin ningún ajuste, nuestro modelo base phi-1 entrenado en *CodeTextbook* logra un rendimiento de HumanEval del 29 % con un modelo de solo 1,3B de parámetros. El modelo más pequeño anterior que logra cerca del 30 % de rendimiento en HumanEval fue Replit-Finetuned con 2,7B de parámetros, que fue entrenado con 100 veces más tokens de entrenamiento que nosotros [Rep23]. Además de esto, ajustar nuestro conjunto de datos *CodeExercises* para obtener phi-1 no solo nos brinda nuestro rendimiento máximo del 51 % en HumanEval, sino que también desbloquea más capacidades de codificación inesperadas (consulte la Sección 3).

Como se menciona 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 conjuntos de datos basados en web (por ejemplo, StackOverflow y CodeContest [LCC+22]), sostenemos que estas fuentes no son óptimas para enseñar al modelo cómo razonar y planificar algorítmicamente. Por otro lado, nuestra arquitectura de modelo y nuestros métodos de entrenamiento son bastante convencionales (Sección 2.3), por lo que dedicamos esta sección principalmente a explicar cómo seleccionamos nuestros datos.

Los conjuntos de datos de códigos estándar [KLA+22, LCC+22] forman un corpus grande y diverso que cubre una amplia gama de temas y casos de uso. Sin embargo, basándonos en la inspección manual de muestras aleatorias, observamos que muchos de estos fragmentos no son muy instructivos para aprender los conceptos básicos de la codificación y adolecen de varios inconvenientes:

- Muchas muestras no son independientes, lo que significa que dependen de otros módulos o archivos externos al fragmento, lo que las hace difíciles de entender sin contexto adicional.
- Los ejemplos típicos no implican ningún cálculo significativo, sino que consisten en código trivial o repetitivo, como la definición de constantes, el establecimiento de parámetros o la configuración de elementos GUI.
- Las muestras que contienen lógica algorítmica a menudo están enterradas dentro de funciones complejas o mal documentadas, lo que hace que sea difícil seguirlas o aprender de ellas.
- Los ejemplos están sesgados hacia ciertos temas o casos de uso, lo que resulta en una distribución desequilibrada de conceptos y habilidades de codificación en todo el conjunto de datos.

Uno solo puede imaginar lo frustrante e ineficiente que sería para un estudiante humano intentar adquirir habilidades de codificación a partir de estos conjuntos de datos, ya que tendría que lidiar con mucho ruido, ambigüedad e información incompleta en los datos. Nuestra hipótesis es que estos problemas también afectan el rendimiento de los modelos de lenguaje, ya que reducen la calidad y cantidad de la señal que asigna el lenguaje natural al código. Conjeturamos que los modelos de lenguaje se beneficiarían de un conjunto de entrenamiento que tenga las mismas cualidades que lo que un humano percibiría como un buen “libro de texto”: debe ser claro, autónomo, instructivo y equilibrado.

En este trabajo, abordamos este desafío directamente y mostramos que, al seleccionar y generar intencionalmente datos de alta calidad, podemos lograr resultados de vanguardia en tareas de generación de código con un modelo mucho más pequeño y menos computo que los enfoques existentes. Nuestra formación se basa en tres conjuntos de datos principales:

- Un datast de filtered code-language, que es un subconjunto de The Stack y StackOverflow, obtenido mediante el uso de un clasificador basado en modelos de lenguaje (que consta de aproximadamente 6 mil millones de tokens).
- Un Dataset Synthtic textbook que consta de <1.000 millones de tokens de libros de texto de Python generados por GPT-3.5.
- Un pequeño dataset synthetic exercises que consta de∼180M de tokens de ejercicios y soluciones de Python.

Describimos esos conjuntos de datos con más detalle en las siguientes subsecciones. En conjunto, los datasets anteriores contienen menos de 7B de tokens. Nos referimos a la combinación de datasets filtered code-language y synthetic textbook como "CodeTextbook" y usarlos en la fase de preentrenamiento para obtener nuestro modelo phi-1-base este modelo ya logra un rendimiento competitivo de HumanEval del 29%. Luego usamos el dataset de 180M token synthetic exercises, denominado "CodeExercises", para ajustar nuestro modelo phi-1-base para obtener phi-1. A pesar del pequeño tamaño del conjunto de datos “CodeExercises”, el ajuste fino de este conjunto de datos es crucial no sólo para grandes mejoras en la generación de funciones simples de Python como se muestra en la Figura 2.1, sino de manera más amplia para desbloquear muchas capacidades emergentes interesantes en nuestro modelo phi-1que no se observan phi-1- base (ver Sección 3).

2.1 Filtrado de conjuntos de datos de código existentes utilizando un clasificador basado en transformador

Comenzamos con conjuntos de datos de código Python disponibles públicamente: utilizamos el subconjunto Python de la versión de duplicada de The Stack y StackOverflow, que juntos contienen más de 35 millones de archivos/muestras, con un total de más de 35B de tokens. Anotamos la calidad de un pequeño subconjunto de estos archivos (alrededor de 100.000 muestras) usando GPT-4: dado un fragmento de código, se le pide al modelo "determinar su valor educativo para un estudiante cuyo objetivo es aprender conceptos básicos de codificación".

Luego usamos este conjunto de datos anotado para entrenar un clasificador random forest que predice la calidad de un archivo/ muestra utilizando su salida incrustada de un modelo codegen previamente entrenado como características. Observamos que, a diferencia de GPT-3.5, que usamos ampliamente para generar contenido sintético (que se analiza a continuación), usamos GPT-4 mínimamente solo para anotaciones sobre la calidad de un pequeño subconjunto de muestras de The Stack y StackOverflow. Por lo tanto, consideramos nuestro uso de GPT-4 simplemente como una forma de evitar tediosos esfuerzos de anotación humana [DLT+23].

Educational values deemed by the filter

High educational value

```python
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(x, 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

```python
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 = ''
        ...
```