

DataFlow：一种由大语言模型驱动的统一数据准备与 workflow 自动化框架，适用于以数据为中心的人工智能时代

DataFlow: 一种由大语言模型驱动的统一数据准备与 workflow 自动化框架, 适用于以数据为中心的人工智能时代

Hao Liang^{*,†}, Xiaochen Ma^{*,†}, Zhou Liu^{*,†}, Zhen Hao Wong^{*}, Zhengyang Zhao^{*}, Zimo Meng^{*}, Runming He^{*}, Chengyu Shen^{*}, Qifeng Cai^{*}, Zhaoyang Han^{*}, Meiyi Qiang^{*}, Yalin Feng^{*}, Tianyi Bai^{*}, Zewei Pan, Ziyi Guo, Yizhen Jiang, Jingwen Deng, Qijie You, Peichao Lai, Tianyu Guo, Chi Hsu Tsai, Hengyi Feng, Rui Hu, Wenkai Yu, Junbo Niu, Bohan Zeng, Ruichuan An, Lu Ma, Jihao Huang, Yaowei Zheng, Conghui He, Linpeng Tang, Bin Cui, Weinan E, Wentao Zhang[‡]

¹Peking University, ²Institute for Advanced Algorithms Research, Shanghai, ³OriginHub Technology, ⁴OpenDataLab, Shanghai Artificial Intelligence Laboratory, ⁵LLaMA-Factory Team

大语言模型 (LLMs) 对高质量数据的快速增长需求, 加剧了对可扩展、可靠且语义丰富的数据准备流水线的需求。然而, 当前实践仍主要依赖于临时脚本和松散定义的工作流, 缺乏原则性的抽象, 阻碍了可复现性, 并在模型参与的数据生成方面支持有限。为解决这些挑战, 我们提出 DataFlow——一种统一且可扩展的由大模型驱动的数据准备框架。DataFlow 采用系统级抽象, 实现模块化、可重用且可组合的数据转换, 并提供类似 PyTorch 风格的流水线构建 API, 用于构建可调试和可优化的数据流。该框架包含近 200 个可重用算子, 以及覆盖文本、数学推理、代码、Text-to-SQL、Agentic RAG 和大规模知识提取六个领域通用的流水线。为进一步提升易用性, 我们引入 DataFlow-Agent, 通过算子合成、流水线规划和迭代验证, 自动将自然语言规范转化为可执行流水线。在六个代表性应用场景中, DataFlow 均持续提升了下流大模型性能。我们的数学、代码和文本流水线优于精心筛选的人工数据集和专用合成基线, 在 Text-to-SQL 上相较 SynSQL 实现最高 +3% 的执行准确率提升, 在代码基准上平均提升 +7%, 在 MATH、GSM8K 和 AIME 上取得 1-3 个百分点的增益。此外, 由 DataFlow 生成的统一 10K 样本数据集使基础模型的表现超越使用 1M Infinity-Instruct 数据训练的对应模型。这些结果表明, DataFlow 为可靠、可复现且可扩展的大模型数据准备提供了实用且高性能的基础架构, 并为未来以数据为中心的人工智能发展建立了系统级基础。

*Equal Contribution, [†]Project Leader, [‡]Corresponding author

✉ Correspondence : wentao.zhang@pku.edu.cn

🔗 Source Code : <https://github.com/OpenDCAI/DataFlow>

📁 Dataset : <https://huggingface.co/datasets/OpenDCAI/dataflow-instruct-10k>

📖 Codebase Documentation : <https://opendcai.github.io/DataFlow-Doc/>

Contents

| | | |
|-------|--------------------------------|----|
| 1 | 引言 | 5 |
| 2 | 背景与相关工作 | 7 |
| 2.1 | 大模型开发中的数据 | 7 |
| 2.2 | 大模型的数据准备 | 7 |
| 2.3 | 现有的大语言模型数据准备系统 | 7 |
| 3 | 数据流系统概述 | 8 |
| 3.1 | 目标与设计哲学 | 8 |
| 3.2 | 系统范围与定位 | 9 |
| 3.3 | 系统工作流 | 9 |
| 4 | 框架设计与架构 | 10 |
| 4.1 | 全球存储抽象与操作符交互 | 10 |
| 4.2 | 分层编程接口 | 10 |
| 4.2.1 | 大语言模型服务 API | 11 |
| 4.2.2 | 操作员编程接口 | 11 |
| 4.2.3 | 提示模板界面 | 12 |
| 4.2.4 | 流水线组合接口 | 13 |
| 4.3 | 操作员分类 | 13 |
| 4.4 | 数据流生态系统 | 15 |
| 5 | 数据流智能体 | 15 |
| 5.1 | 智能体角色 | 16 |
| 5.2 | 智能流水线推荐 | 16 |
| 5.3 | 概要 | 17 |
| 6 | 用例 & 流水线 | 17 |
| 6.1 | 案例研究: DATAFLOW 中的文本到 SQL 数据流水线 | 17 |
| 6.1.1 | 运算符 | 17 |
| 6.1.2 | 流水线 | 19 |
| 6.1.3 | DATAFLOW 支持机制 | 19 |
| 7 | 实验 | 20 |
| 7.1 | 文本数据准备 | 20 |
| 7.1.1 | 实验情景 | 20 |
| 7.1.2 | 实验结果 | 21 |
| 7.2 | 数学推理数据准备 | 21 |
| 7.2.1 | 实验情景 | 22 |
| 7.2.2 | 实验结果 | 22 |

| | | |
|-------|--------------------------|----|
| 7.3 | 代码数据准备 | 23 |
| 7.3.1 | 实验情景 | 23 |
| 7.3.2 | 实验结果 | 23 |
| 7.4 | 文本到 SQL 的数据准备 | 24 |
| 7.4.1 | 实验情景 | 24 |
| 7.4.2 | 实验结果 | 25 |
| 7.5 | AgenticRAG 数据准备 | 25 |
| 7.5.1 | 实验情景 | 26 |
| 7.5.2 | 实验结果 | 26 |
| 7.6 | 知识提取 | 27 |
| 7.6.1 | 实验情景 | 27 |
| 7.6.2 | 实验结果 | 28 |
| 7.7 | 通过 DataFlow 实现统一的多领域数据准备 | 28 |
| 7.7.1 | 实验情景 | 28 |
| 7.7.2 | 实验结果 | 29 |
| 7.8 | Agentic 协调 | 30 |
| 7.8.1 | 实验情景 | 30 |
| 7.8.2 | 实验结果 | 31 |
| 8 | 结论 | 31 |
| A | 作者贡献 | 39 |

1 引言

大语言模型（LLMs）已迅速从研究原型演变为自然语言处理及其他领域的基础架构。自 OpenAI 通过大规模人工标注推出 GPT [1] 系列并开启大语言模型（LLMs）时代以来，缩放定律研究 [26, 55] 持续表明，数据质量与数量是决定模型性能的核心因素。

随着模型规模持续增长以及下游任务日益复杂，训练语料库的大小与语义多样性也实现了显著扩展 [29, 67]。现代大模型开发如今依赖于多阶段、语义密集型的数据准备流水线，该流水线整合了合成、精炼、过滤以及跨万亿 token 的领域特定变换 [6, 47, 61]。

然而，尽管高质量数据起着至关重要的作用，大模型的数据准备仍处于碎片化且基本无标准的状态。大多数从业者仍然依赖临时编写的脚本和松散标准化的工作流，这些工作流缺乏显式的数据流抽象、明确定义的原子操作，或任何形式的流水线级最优化。

缺乏统一且可编程的范式，使得流水线在不同项目间难以复现、扩展或比较 [6, 47, 48]。

这一问题随着后训练任务日益精细化的趋势而加剧，例如指令微调、思维链生成或函数调用等任务，其中数据准备中的语义丰富性和语义准确性对于实现精确的任务级模型行为至关重要 [62, 72]。

针对这一碎片化现象，近期出现了多个系统，旨在标准化大语言模型的数据监管。诸如 NeMo Curator [47] 和 Data-Juicer [6] 等框架提供了丰富的功能——包括图像描述生成、重写、分类和多模态处理——显著提升了大规模语料库构建的效率。然而，这些系统本质上仍以数据提取和过滤为导向，其抽象机制对表达具有细粒度语义控制的迭代式、模型在环中的生成工作流支持有限。因此，它们并不适用于以数据合成和多步语义精炼为核心而非辅助环节的工作流。

这一限制正变得日益重要。大模型不再仅仅是数据的消费者，同时也是数据的生产者。由于大规模人工标注成本过高，近期的研究大量采用基于大模型的数据合成工作流，在大规模上构建高质量语料库 [3]。多项近期报告表明，在许多场景下，经过精心合成的数据甚至可以超越高质量筛选的数据表现 [60, 69]，进一步凸显了大模型驱动生成工作流的重要性。

基于这些趋势，我们认为，大语言模型数据准备必须建立一个统一框架，将大语言模型驱动的数据合成提升为第一类、可编程的数据流抽象。该框架应满足以下要求：(1) 提供细粒度、可组合的算子，用于模型在环中的生成与语义精炼；(2) 支持显式、可验证的流水线定义，作为可检查、领域无关的开源协议，用于大语言模型数据准备——正如 `torch.nn.Module` 之于深度学习中的模型组合标准化；(3) 保持后端无关性，以集成不同的大语言模型引擎和存储后端；(4) 支持在不同模型、任务和领域间进行有原则的工作流组合、复用与最优化，同时进一步支持智能体驱动的自动工作流构建。综上所述，这些需求标志着数据准备范式的转变——从事后语料清洗转向以大语言模型为中心的工作流，通过迭代合成与精炼，构建高保真度、语义丰富且任务对齐的合成语料库。

受这一趋势的启发，我们提出了 DATAFLOW，这是一个统一且自动化的端到端大模型驱动的数据准备框架，其中包含一个 DATAFLOW-智能体，使用户能够直接从自然语言规范中组合工作流。DATAFLOW 将大模型置于操作符生态的核心位置：大多数操作符由大模型驱动，仅有少量通过启发式规则或小型模型实现。该框架提供了超过 180 个操作符，分为生成、评估、过滤和精炼四类，并包含 90 多个可复用的提示模板，支持操作符级别的组合以及跨任务的一致行为。利用这些基础组件，DATAFLOW 包含一系列前沿（SOTA）合成工作流，涵盖数学推理、原始文本、代码、文本转 SQL、Agentic RAG 风格数

据, 以及从网页或 PDF 文档中大规模抽取问答数据。所有工作流均在 DATAFLOW 的通用抽象下表达, 无需特定任务的粘合代码, 遵循生成-评估-过滤的工作流, 并辅以针对性的精炼阶段。

为确保可用性、可扩展性和长期可维护性, DATAFLOW 采用类似 PyTorch 的编程接口, 通过模块化的 Python 类和函数暴露其核心抽象、全局存储、大语言模型服务、操作符、提示模板以及流水线。这种以代码为中心的设计避免了复杂的 YAML 或基于 shell 的配置方案, 提供了面向 IDE 的开发工作流, 包括代码补全和可靠的导航功能。除了核心库之外, 操作符、提示模板和流水线可在主仓库外部独立开发, 并打包为独立的 Python 模块, 使从业者能够将领域特定的组件作为一级 DATAFLOW-EXTENSIONS 发布和复用。为支持这一生态系统, DATAFLOW 内置一套命令行界面 (CLI) 工具链, 可从操作符模板到完整流水线仓库, 快速搭建新的扩展包, 标准化开发实践并降低社区贡献的门槛。最后, DATAFLOW-AGENT 作为 Agentic 协调层, 能够将自然语言规范转化为可执行的流水线, 并在需要时自动合成与调试新操作符, 进一步加速构建可扩展且语义丰富的大语言模型驱动的数据准备工作流。

在六个由 DATAFLOW 实现的流水线上的大量实验表明, 我们的设计哲学在多种数据准备情景中均有效, 能够持续生成高质量的训练数据。在所有情景下, 所生成的 DATAFLOW 语料库表现达到甚至超越当前最先进 (SOTA) 基准, 包括经过精心筛选的人工标注语料库、专用的合成工作流以及强大的 Qwen2.5-Instruct 系列。例如, DATAFLOW 合成的数学推理数据在 MATH、GSM8K 和 AIME 测试集上相较于高质量合成基准提升 1-3 个百分点 [28, 43]; 我们的文本转 SQL 流水线相较于包含 250 万样本的 SynSQL 语料库 [37], 在使用少于 0.1M 训练样本的情况下, 执行准确率提升超过 +3%; 而基于 DATAFLOW 的代码流水线相较于广泛使用的公开代码指令数据集, 平均提升超过 7% [5, 63]。

此外, 通过将 DATAFLOW 生成的文本、数学和代码数据整合到一个统一的语料库 DATAFLOW-INSTRUCT-10K 中, 我们发现仅使用 10K 个样本进行训练, Qwen2-base 和 Qwen2.5-base 就能超越在 1M Infinity-Instruct [39] 个样本上训练的模型, 同时接近其对应版本的 Qwen-Instruct 模型性能。这表明 DATAFLOW 能够生成高质量且领域多样的监督信号, 从而在数据效率方面带来显著提升。

这些结果共同表明, DATAFLOW 不仅是一个基于大语言模型的数据准备端到端系统, 还是一套全面的操作符与算法库, 以及一个开放且用户友好的协议框架。基于六个最先进的模板流水线和大量可重用操作符构建, DATAFLOW 为以大语言模型为中心的数据构建提供了统一基础, 支持原则性强、语义丰富且可扩展的工作流, 从而在各个领域提升了可编程性、可复现性和数据质量。

总体而言, 我们的主要贡献总结如下:

- **一种统一的基于大模型的数据准备框架。**我们提出 DATAFLOW, 这是一个基于可组合抽象和以大模型为核心的算子执行模型的统一的大模型数据准备系统。
- **丰富且可扩展的算子-流水线生态系统。**DATAFLOW 提供了近 200 个可复用的算子以及六种覆盖文本、数学推理、代码、文本转 SQL、Agentic RAG 数据和大规模 QA 抽取的 SOTA 流水线。
- **面向开发者和开源社区的编程模型。**通过类 PyTorch 的 API、IDE 原生工具以及基于 Python 包的插件式扩展, DATAFLOW 支持可复现的实验、便捷的定制化, 以及由社区驱动的扩展, 共同构建 DATAFLOW-生态体系。
- **用于自动化流水线构建的智能体编排层。**DATAFLOW-AGENT 从自然语言意图生成可执行的流水线, 降低了构建可扩展且语义丰富的大模型驱动工作流的门槛。

- **广泛的实验验证与开源数据发布。**六条流水线的实验表明，DATAFLOW 生成的数据能够持续提升下游大模型的性能和数据效率。我们额外发布了一个完全使用 DATAFLOW 生成的高质量、多领域数据集，以支持后续研究与基准测试。

2 背景与相关工作

2.1 大模型开发中的数据

大模型的发展涉及多个关键阶段，其中训练尤为关键，因为模型能够从大规模语料中学习基本的语言模式。在此阶段，模型会接触到来自各个领域的大量文本数据，从而获得对语言的广泛理解。

因此，训练数据的质量和多样性直接影响模型在不同上下文中的有效泛化能力 [19, 38]。最近，大语言模型的快速发展带来了训练数据量的显著增加 [1, 57]。在这种情况下，数据的质量和数量变得更加至关重要。

高质量的数据可以显著提升模型性能 [44]。随着数据量的增加，确保高质量数据变得更加困难，因为这需要额外的资源进行数据清洗、选择和标注 [3]。低质量的数据可能导致模型学习到错误的模式并产生不准确的预测。此外，数据多样性不足可能导致模型在特定领域表现良好，但在跨领域任务中表现出较差的泛化能力。同时，数据分布的变化可能加剧模型对训练分布的过度依赖，降低其在真实场景中的适用性。

2.2 大模型的数据准备

如上所述，数据准备是训练大模型的关键步骤，对模型的性能和泛化能力具有显著影响。随着大模型规模的持续扩大，数据准备的复杂性和效率已成为关键的研究焦点。然而，尽管 Apache Spark [71]、Dask [52] 和 Hadoop [13, 20, 65] 等系统在大规模抽取-变换-加载 (ETL) 方面功能强大，但它们并不适合现代大模型的数据准备。这些框架原则上可以通过调用大模型或嵌入模型作为用户自定义函数来执行语义清洗，但它们不原生支持模型环处理、GPU 高效的批量处理或基于 token 的文本操作。更重要的是，其内置算子专注于结构化数据，对非结构化文本的功能支持非常有限，这意味着诸如分词、语言检测、文档分割、语义去重或安全过滤等关键步骤必须通过临时编写的用户自定义函数手动实现。这导致了巨大的开销和工程复杂性，使得通用的大数据引擎无法满足大模型语料库构建所需的规模化、语义密集型流水线需求。

基于大模型的方法已被广泛应用于数据质量评估和数据选择。例如，MoDS [15] 利用 DeBERTa 进行得分并保留高质量数据，而 Alphagassus [7] 使用 ChatGPT 评估数据准确率。其他研究则采用 GPT-4 进行数据重写和质量提升。更多详细信息，请参阅关于大模型的综述数据 [3]。

2.3 现有的大语言模型数据准备系统

近期的研究越来越多地将大语言模型训练数据准备视为一个首要的系统问题。表 1 总结了主要框架的特征区别。

NeMo Curator [47] 是由 NVIDIA 提供的开源、GPU 加速库，提供模块化流水线，用于大规模大语言模型的数据监管，包括数据下载与提取（例如 Common Crawl、arXiv、Wikipedia）、语言识别、文本清

Table 1 现有大模型数据准备系统的高层次对比。

| Dimension | Data-Juicer [6] | NeMo Curator [47] | DataFlow (ours) |
|-------------------|------------------------|----------------------------|--------------------------------------|
| Primary focus | Filtering / Cleaning | Large-scale Curation | LLM-driven Synthesis + Refinement |
| Programming model | Config-based Recipes | Component-based Pipelines | PyTorch-like Operators & Pipelines |
| LLM integration | Partial (some gen ops) | Minimal (mainly filtering) | First-class Serving + Templates |
| Automation | Recommendation Agent | None | Pipeline Construct/Debug Agent |
| Extensibility | OperatorZoo / Cookbook | Custom Scripts | Extension Packages + CLI Scaffolding |

洗、基于启发式和学成的质量过滤、领域与毒性分类、文档级和语义级去重、隐私过滤，甚至合成数据生成，全部基于 Dask/RAPIDS 构建，并设计为可扩展至多节点、多 GPU 环境。

Data-Juicer [6] 是一个“一站式”数据处理系统，将大语言模型数据配方抽象为可组合的算子：原系统已提供 50 多个算子用于构建和评估文本数据混合，而 2.0 版本将其扩展至 100 多个算子，涵盖文本、图像、视频和音频，支持分析、清洗、合成、标注以及后训练数据流水线，并与 Ray 和 Huggingface Datasets 紧密集成。

这些系统显著提升了大语言模型数据准备的效率和质量，但它们仍主要作为以配置为中心的工具包。相比之下，我们的框架围绕一个包含近 200 个可重用文本专用算子的丰富库构建，能够对清洗、变换、合成和评估实现细粒度控制；从这些算子实例化出的多个流水线能持续带来显著的下游性能提升，甚至不同流水线生成的数据简单混合后仍保持高度有效性。此外，该系统采用模块化、类似 PyTorch 的“构建块”设计，具有轻量且定义清晰的接口，使得数据智能体能够自然地以编程方式组合、编排和调用数据处理流水线。

3 数据流系统概述

在本节中，我们介绍 DATAFLOW，这是一个统一且自动化的系统，用于标准化和简化大模型的多领域数据准备。

3.1 目标与设计哲学

DataFlow 围绕六个核心目标设计：

使用便捷性。 一个受 PyTorch 启发的 [49]、IDE 友好的编程接口，使用户能够以最少的样板代码构建和调试复合数据准备流水线。

可扩展性。 遵循类似于 `torch.nn.Module` 的模块化抽象，新的算子和算法可以作为即插即用的组件添加，并与现有工作流自然组合。

统一范式。 DATAFLOW 在标准化抽象层下统一了异构数据准备工作流。该设计在确保一致性与可复现性的 标准化 与各领域所需的 定制化 之间取得平衡，从而实现流水线的高效复用与适应。

性能效率。 DATAFLOW 中的官方流水线实现了与当前最优数据准备方法相当或更优的性能，表明合一不会带来显著开销。

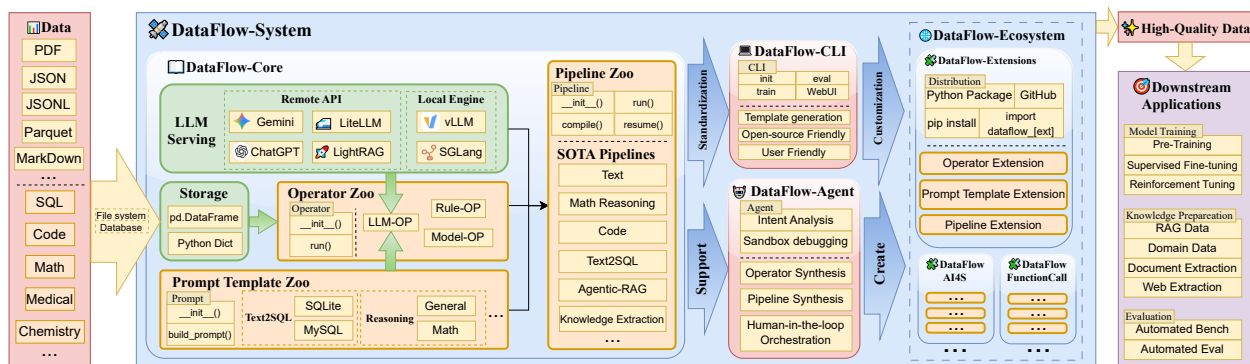


Figure 1 DATAFLOW 的高层架构。该系统由核心执行引擎（存储、算子、模板和大模型服务）、可复用的流水线、面向用户的控制层（命令行界面和智能体），以及用于领域专用工作流的可扩展生态系统组成。DATAFLOW 生成高质量、任务对齐的数据集，供下游大模型应用使用。

智能自动化。一个轻量级的 Agentic 子系统利用核心抽象，解析自然语言意图，并自动构建或调整操作符和流水线，支持快速原型设计并减少手动工程工作。

开源范式。DATAFLOW 的目标是作为大语言模型数据准备的社区标准。其统一的抽象能够实现可复现的流水线共享、大语言模型后端的透明切换以及受控的实验。

3.2 系统范围与定位

DATAFLOW 覆盖了以大语言模型为中心的数据准备全流程。如图 1 所示，其核心系统为存储、大语言模型服务、操作符、提示模板和流水线提供了统一的抽象——定义了所有变换执行的运行基础。在核心之上，两个面向用户的控制层，命令行接口（CLI）和 DATAFLOW-智能体，支持可脚本化和自动化的流程构建。在引擎之外，DATAFLOW-扩展提供了一个模块化接口，用于添加基于 Python 包的操作符、模板和流水线。基于该接口构建的领域专用包共同构成了更广泛的 DATAFLOW-生态。这些组件共同定义了系统的边界：DATAFLOW 提供数据准备的抽象和控制层，而下游的大语言模型训练、评估和检索应用则消费其输出。

3.3 系统工作流

图 1 还展示了 DATAFLOW 的端到端工作流。系统从常见的文件格式（例如，JSON、JSONL、CSV、Parquet、Markdown、PDF）以及领域特定的数据源（如 SQL 日志和代码仓库）中摄入数据，将所有输入转换为由核心存储层维护的统一表格表示。操作符通过共享存储读取和写入中间结果，从而在变换阶段之间实现一致的数据流。

操作符实现诸如生成、精炼、过滤和评估等变换。由大语言模型驱动的操作符通过统一的服务抽象调用本地推理引擎（例如，vLLM [35]、SGLang [73]）或基于在线 API 的服务（例如，Gemini [56]、ChatGPT [1]），而基于规则和小模型的操作符则独立于大语言模型后端执行。

Pipeline Zoo 中的流水线将这些算子组合成可重复使用的工作流，适用于文本合成、数学推理、代码处理、文本转 SQL 生成、Agentic RAG 以及大规模知识提取等任务。流水线可直接执行、编译以实现优化执行、从中间状态恢复，或适应新领域。

用户通过命令行界面（CLI）或 DATAFLOW-智能体与 DATAFLOW 交互以执行工作流：CLI 发出明确的执行指令，而智能体则将自然语言规范转化为可执行的工作流并进行迭代调试。工作流的输出为高质量、任务对齐的数据集，可无缝集成至下游大语言模型应用中。

4 框架设计与架构

本节介绍了 DATAFLOW 的内部设计，并在第 3 节中形式化了其抽象所基于的执行模型。DATAFLOW 围绕四个架构支柱构建：(1) 一种全局存储抽象，用于维护数据集的正则表格表示，并协调所有数据访问；(2) 针对 LLM 服务、操作符、提示模板和流水线的分层编程接口集合；(3) 一种合理的操作符分类方案，能够调和开放领域的领域需求与一组紧凑的可重用变换原语之间的矛盾；以及 (4) 一种扩展机制，支持不断增长的用户贡献组件生态系统。这些要素共同构成了一个可扩展且可扩展的底层平台，用于构建、执行和共享以 LLM 为中心的数据准备工作流。

4.1 全局存储抽象与操作符交互

DATAFLOW 执行基础的核心是一个统一的存储抽象，它维护数据集的正则表格表示，并在工作流执行期间协调所有数据访问。面向大模型的数据——如指令、响应、思维链迹、得分和元数据——自然地以与每个样本相关的键值字段形式表达，使得表格结构成为一种合适且富有表现力的组织格式。存储层将数据管理与算子逻辑解耦，通过 `DataFlowStorage` 基类暴露一个最小化且与后端无关的 API。这种设计允许自定义存储后端——如文件系统、对象存储或数据库实现——被集成，而无需改变算子的行为。

抽象提供了两种主要操作：

- `read()`：以操作符所需格式检索当前数据集（或相关字段）。
- `write(data)`：更新或追加字段到共享数据集表示。

通过这些操作集中所有访问，可确保操作员对物理存储布局保持无感知，同时一个操作员产生的中间产物可立即被其他操作员使用。典型的操作员交互遵循图 2 中的模式。

```
def run(self, storage: DataFlowStorage, **kwargs):  
    inputs = storage.read()           # 1. Read input  
    results = operator_transform(inputs, **kwargs) # 2. Transform the data  
    storage.write(results)            # 3. Write output
```

Figure 2 在 DATAFLOW 中，算子的 `run()` 方法的标准执行模式。在 `run()` 内部，算子通过 `storage.read()` 从全局 `DataFlowStorage` 读取输入，应用其变换逻辑，并通过 `storage.write()` 将更新后的字段写回。这一读取-变换-写入模式捕捉了数据在整个工作流中从一个算子流向下一个算子的方式。

由于操作符仅针对此逻辑抽象进行操作，因此可以在不修改其内部结构的情况下重新排序、重新组合或批量处理，且对存储后端的改进（例如添加分布式或基于数据库的实现）无需更改操作符级别。默认的存储实现使用 Pandas 作为执行基础，并支持常见的输入/输出格式，如 JSON、JSONL、CSV 和 Parquet。

4.2 分层编程接口

DATAFLOW 暴露了一个围绕四个核心抽象构建的分层编程接口。(1) 服务接口为跨异构后端发出大语言模型推理请求提供了一种统一机制。(2) 操作符定义可重用的数据变换单元，当需要大语言模型驱动的计算时，可选择性调用服务层。(3) 提示模板指定如何将操作符输入渲染为具体的提示，并规定模型

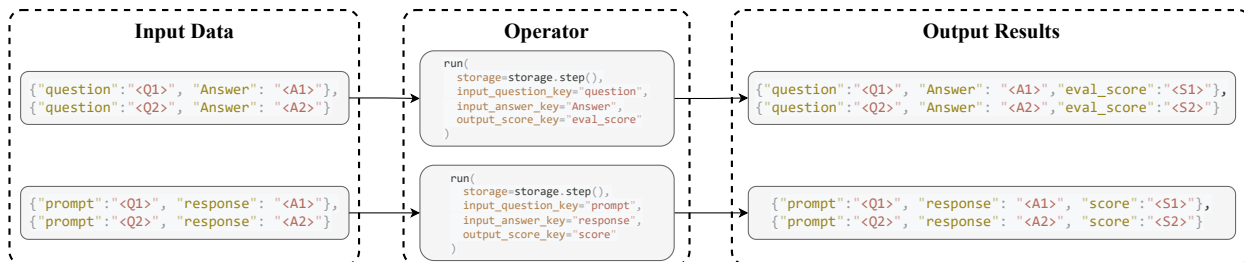


Figure 3 操作符的 `run()` 方法通过基于键的绑定与数据交互的示例。这种灵活的键绑定机制无需预处理即可适应任意数据集，并支持操作符的无缝组合。

输出应如何结构化或受限，为一致的提示构造提供声明式接口。(4) 流水线将操作符组合成具有显式数据依赖关系的多阶段工作流，并支持可选的编译以进行验证和最优化。以下小节将详细描述这些抽象。

4.2.1 大语言模型服务 API

LLM 驱动的操作符依赖于一个统一的服务 API，该 API 抽象了异构的模型后端。该 API 提供了一个单一的高层入口点，`generate_from_input(user_inputs, system_prompt, json_schema)`，接收由调用操作符组装的提示列表，并返回模型生成的输出列表。可选参数如 `system_prompt` 或输出 `json_schema` 可在需要时启用结构化提示和解码。此接口使操作符免受后端特定考虑因素的影响，例如批量处理、重试策略、请求路由和速率限制。

服务层支持以下两者：

- 本地推理引擎（例如，vLLM [35]、SGLang [73]），利用后端级别的并行性实现高吞吐量执行；
- 基于在线 API 的服务（例如，ChatGPT [1]，Gemini [56]），对于此类服务，DATAFLOW 采用多线程请求分发以最大化吞吐量。

这种统一的服务抽象降低了基于大语言模型的操作符实现负担，并支持灵活的后端置换，从而便于评估不同大语言模型选择对数据准备质量的影响。

4.2.2 操作员编程接口

操作符是 DATAFLOW 中的基本变换单元。它们遵循两阶段接口，将初始化与执行清晰地分离：初始化配置操作符，而执行则完成变换。这种分离使得异构行为（从基于大模型的生成到基于规则的过滤）能够以统一的抽象形式表达。

在初始化阶段（`__init__()`），一个算子会接收诸如超参数或任务特定设置之类的配置参数。基于大模型的算子在此阶段可能还会绑定一个大模型服务对象和一个提示模板对象，而基于规则和轻量级模型的算子则完全省略这些绑定。因此，初始化阶段捕获了所有静态配置和外部依赖，使执行阶段能够专注于数据转换。

一个算子的 `run()` 方法实现了其变换逻辑，并构成了流水线中的执行单元。为了保持算子的通用性和易于组合性，`run()` 仅接受一个 `DataFlowStorage` 对象以及一组 `input_*` 和 `output_*` 键。将这些键解释为键值对时，`input_*` 键表示要读取作为输入字段的存储列，而 `output_*` 键则表示每个处理数据

```

class TranslatePipeline(PipelineABC):
    def __init__(self):
        super().__init__()
        # Init Resources
        self.storage = FileStorage(
            entry_file="input_data.jsonl",
        )
        self.llm_serving = APILLMServing(
            api_url="<api_url>",
            model_name="gpt-4o",
        )
        # Initialize Operators
        self.op1 = PromptedGenerator(
            llm_serving=self.llm_serving,
            system_prompt="Translate the content to Chinese",
        )
        self.op2 = PromptedGenerator(
            llm_serving=self.llm_serving,
            system_prompt="Translate the content to English",
        )

    # forward function of the Pipeline
    def forward(self):
        # execute operators
        self.op1.run(
            self.storage.step(),
            input_key='raw_content',
            output_key='content_CN'
        )
        self.op2.run(
            self.storage.step(),
            input_key='raw_content',
            output_key='content_EN'
        )

if __name__ == "__main__":
    TransPipeline = TranslatePipeline()
    Transpipeline.compile() # Optional
    # excute pipeline, resume from `op2`
    Transpipeline.forward(resume_step=1)

```

Figure 4 DATAFLOW 流水线 API 示意图。示例展示了流水线如何声明其存储和服务后端，使用特定任务配置实例化算子，并通过 `forward()` 及输入/输出键绑定执行它们。该接口支持编译和逐步恢复，可实现灵活且模块化的工作流构建。

项要写入的新列名称。图 3 揭示了这一映射关系。该设计提供了灵活的输入输出绑定，能够自然适应多样的上游数据集，同时声明的键形成了算子之间的有向依赖图，从而支持拓扑调度和下游最优化检查。

通过将配置与执行分离，并将状态变化限制为对共享存储的显式基于键的读写操作，操作符抽象保持了轻量、确定性和易于组合的特性。这些特性使得 DATAFLOW 能够在单一、可移植的接口下支持广泛的变换行为，同时在整个系统中保持一致的执行语义。

4.2.3 提示模板界面

提示词是引导大模型执行特定任务变换的主要机制。每个由大模型驱动的操作符都依赖于一个提示词，那些具有相同高层逻辑的算子通常仅在细微的提示词差异上有所不同。例如，在文本转 SQL 生成任务中，为 SQLite 和 MySQL 生成查询所采用的算子逻辑完全相同；唯一的区别在于通过提示词传达的微小语法调整。为了在支持复用的同时适应领域特异性变化，DATAFLOW 通过专用的提示词模板接口，将提示词构建与算子实现解耦。

一个提示模板封装了一个可重用的提示模式，并提供参数化占位符，由操作符在执行时填充。每个由大语言模型驱动的操作符在 `__init__()` 期间初始化其关联的模板，遵循与其他系统组件相同的配置-执行范式。在执行过程中，操作符调用模板的 `build_prompt()` 方法，将与任务相关的信息——如输入字段、模式提示或上下文元数据——组合成一个具体的提示，随后传递给大语言模型服务层。这种封装使得操作符的变换逻辑对提示如何渲染保持无关。

为了便于操作符与模板之间的一对多映射，由大模型驱动的操作符暴露了一个统一的 `op.ALLOWED_PROMPTS` 接口，该接口枚举了所有兼容的提示模板。这种设计使得操作符能够通过简单地切换或调整模板，在不同领域或任务中灵活复用，而无需修改操作符的逻辑。

总体而言，提示模板接口为提示构建提供了一种声明式机制，促进了在紧密相关任务间的算子复用，并确保了 DATAFLOW 的 LLM 驱动工作流中提示行为的一致性。

4.2.4 流水线组合接口

基于上述抽象，DATAFLOW 提供了一个流水线接口，使用户能够将算子组合成多阶段的数据准备工作流。流水线被表示为算子的有序序列（或轻量级有向无环图），形成一个端到端的执行图，以捕获预期的数据流。图 4 展示了流水线 API 及其核心组件。

流水线 API 采用类似 PyTorch [49] 的设计，其中 `__init__()` 方法负责资源分配和算子配置，而 `()` 方法则编码单次执行过程。在 `forward()` 中，特定于算子的键绑定隐式定义了数据流拓扑，使得流水线能够以模块化、可读性强且对 IDE 友好的方式构建。

从功能上讲，流水线接口提供了一个内置的 `compile()` 过程，在执行前对操作符序列进行静态分析。在编译过程中，DATAFLOW 会提取操作符的依赖关系和参数，构建相应的有向无环图 (DAG)，并进行基于键的验证，以检测缺失字段、类型不一致以及格式错误的依赖链。与立即执行操作符不同，`compile()` 将所有操作符配置和依赖信息记录下来，生成一个延迟执行计划。这种延迟构建的设计遵循工厂方法模式 [16]，其中对象的创建与对象的执行相分离：每个操作符的 `run()` 方法的实际调用被推迟到后续的 `forward()` 调用时才执行。

编译后的执行图首先向 DATAFLOW-Agent 提供完整的结构信息，使其能够在一份报告中揭示所有关键项和依赖相关的错误。这显著减少了智能体所需的调试轮次，并降低了相关的推理成本。此外，编译后的图定义了一个最小且高效的执行计划，支持检查点和逐步恢复等高级运行时特性，从而提升了迭代开发效率和大规模流水线构建能力。

4.3 操作员分类

DATAFLOW 中的算子封装了多种多样的数据处理算法，这些算法在组合后可支持端到端的大语言模型数据准备工作流。作为一个旨在服务于任意多个领域的统一且可扩展的框架，DATAFLOW 必须同时容纳无限扩展的领域特定算法，同时提供稳定且易于理解的算子空间。这两种相互冲突的需求——无边界领域要求与概念紧凑性的需求——带来了内在张力。为调和这一矛盾，DATAFLOW 沿着多个正交的分类维度组织算子。每个维度内的类别互斥，而各维度之间则并行存在。该分类方案已在本文涵盖的多样化领域中得到验证，包括超过六种前沿的数据准备流水线，充分证明了其表示能力的充分性与可扩展的普适性。

模态维度 基本的分类将算子按其处理的模态进行分离，例如文本、视觉内容或类似文档的输入。必须区分模态，因为同一模态内的算子具有兼容的输入-输出语义并可以互操作，而不同模态之间的算子通常无法直接组合。DATAFLOW 主要作用于文本表示，非文本模态首先通过特定于模态的算子处理，这些算子将原始输入（如图像或 PDF）解析或转换为文本，之后才应用下游变换。因此，明确的模态分类使这一转换流程变得清晰，并使流水线编译器能够验证算子链，确保模态转移被正确指定，且仅兼容的算子被组合。

核心维度与领域特定维度。 第二个分类区分了核心算子和领域算子。核心算子反映了 DATAFLOW 的基本设计哲学，并作为大多数其他算子的理论基础。尽管领域算子可能包装或专门化核心算子，但它们的语义通常可以通过实例化相应核心算子的参数来表达。核心算子在数量上被有意限制且相对稳定，是新用户的推荐入门点。相比之下，随着新领域、新模态或新任务的出现，领域算子会不断扩展。尽管理

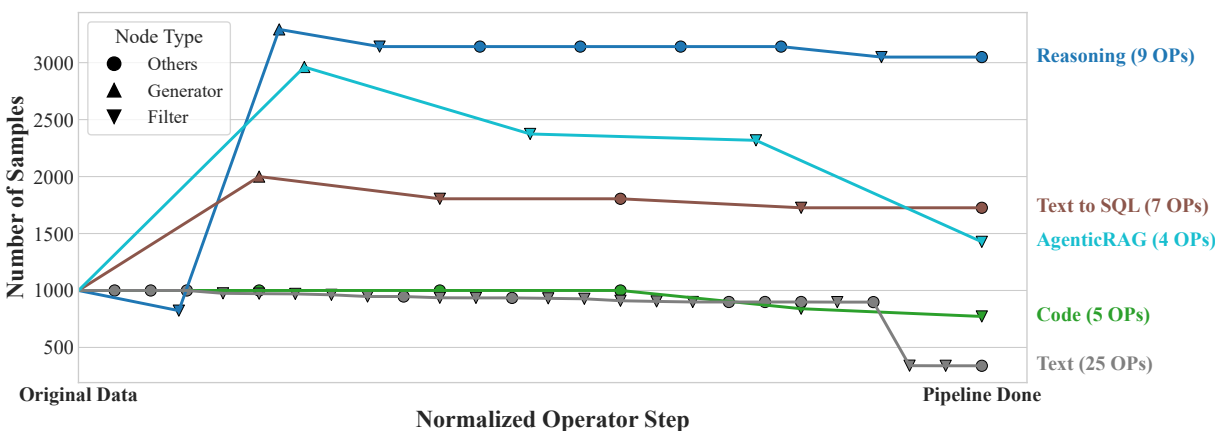


Figure 5 DATAFLOW 流水线中各操作阶段的样本数量演化情况。所有流水线均以 1000 个输入样本开始。文本流水线主要执行预训练数据过滤，代码流水线则专注于基于现有指令数据扩展代码能力；因此，这两条流水线均不涉及任何生成式组件。

论上无边界，但 DATAFLOW 中包含的领域算子仅限于支持现有领域中最优流水线所需的那些，以确保实际简洁性并避免不必要的泛滥。

功能维度 在更细的粒度上，操作符可分为四类功能——generate、evaluate、filter 和 refine——每一类都捕捉了数据准备过程中的独特变换模式。这些类别与 DATAFLOW 作为数据合成框架的核心设计哲学相一致：流水线首先通过生成扩展候选空间，然后对结果进行评分和过滤，期间可选地插入精炼阶段。这一生成-评估-过滤-精炼范式构成了 DATAFLOW 中大多数流水线设计的基础。如图 5 所示，当流水线以 1,000 个输入样本开始时，数据项的数量通常在生成阶段增加，随后在评估、过滤和精炼操作符应用后逐渐减少。

为了使这一范式具体化，DATAFLOW 定义了四类操作符，每类都有明确的语义和命名规范。在本讨论中，我们采用 DATAFLOW 所使用的表格表示法：每一行表示一个数据样本，每个字段对应该样本中的一个命名列。

- **生成**。这些算子通过添加新的文本字段或生成额外的行来扩充数据。以 **Generator** 结尾的算子会向现有行添加新字段，而以 **RowGenerator** 结尾的算子则会增加行数。示例用法包括生成对问题的回答。
- **评估**。这些算子用于为单个样本或整个数据集计算得分或标签。**SampleEvaluator** 算子为每一行附加评估元数据，而 **DatasetEvaluator** 算子则输出数据集级别的度量指标。示例包括为数学问题分配难度等级，或根据主题对问答对进行分类。
- **筛选器**。这些操作符根据现有字段或评估结果推导出的条件来减少行数。它们的语义保持行内容不变，除了新增的评估字段。例如，移除答案错误的样本。
- **精炼**。这些算子修改现有行中的特定字段，而不改变样本数量。它们通常应用轻量级变换，例如从文本中移除网址或表情符号。算子通常以后缀 **精炼器** 结尾。

在这些维度上，DATAFLOW 支持系统性可扩展性和有限的概念复杂度：模态维度和核心-领域维度组织

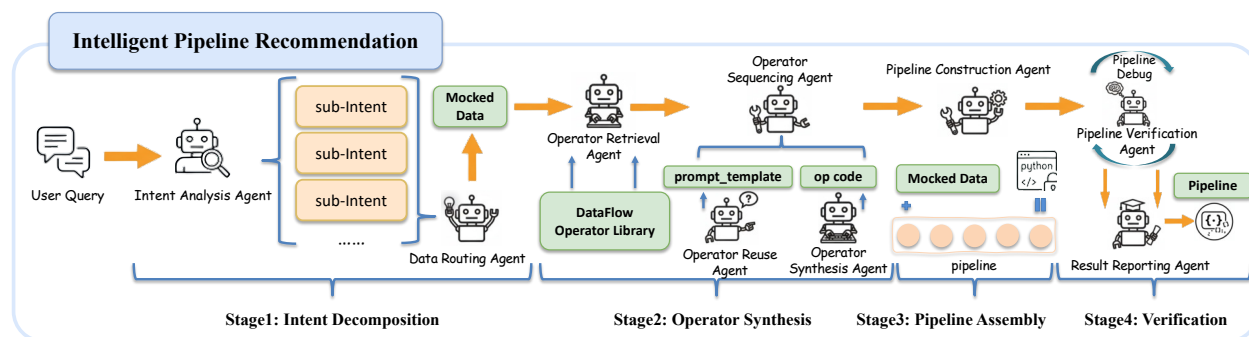


Figure 6 DATAFLOW-AGENT 架构：一种由 LangGraph 驱动的多智能体工作流，可将自然语言意图转化为经过验证的可执行有向无环图流水线。

了一个开放的算子生态系统，而功能维度则提供了一组紧凑且可复用的变换原语，用于构建可扩展的大型语言模型数据准备工作流。

4.4 数据流生态系统

一个统一的数据准备框架必须能够容纳无限扩展的算法和工作流，这自然导致了操作符和流水线的无界空间。为了以可维护的方式组织这种可扩展性，DATAFLOW 引入了 DATAFLOW-Extension 的概念：一种模块化包，用于封装额外的操作符、提示模板和流水线。用户贡献的扩展共同构成了更广泛的 DATAFLOW-Ecosystem，这是一个类似于 Python 包生态系统的即插即用环境，使从业者能够轻松发布、共享和重用领域特定的组件。

为了简化扩展开发，DATAFLOW 通过 DATAFLOW-CLI 提供了自动化项目脚手架功能。根据少量高层次的规格说明，CLI 会生成可直接使用的算子、提示模板、流水线模板，甚至适用于通过 PyPI 或 GitHub 发布的完整仓库布局。开发者只需在这些生成的框架中实现特定任务的逻辑即可。核心系统和扩展包均可通过 Python 包管理器安装并导入，而懒加载机制确保多个扩展能够共存，对环境干扰极小。

除了命令行界面 (CLI) 之外，DATAFLOW-AGENT 还支持通过自然语言驱动的方式构建算子和流水线。利用嵌入在大型语言模型中的领域知识，该智能体能够合成有效的数据变换逻辑，并自动化常见的设计步骤，显著降低了编写高质量 DATAFLOW-扩展的成本。

DATAFLOW-CLI 和 DATAFLOW-AGENT 共同降低了扩展开发的开销，并促进了社区驱动的发展。我们的目标是培育一个可持续发展的开源生态系统，使得基于标准化操作符、提示模板和流水线构建的数据准备配方能够被共享、复现和改进，从而最终加速数据驱动机器学习社区的整体进展。

5 数据流智能体

DATAFLOW-AGENT 作为 DATAFLOW 框架之上的智能编排层。它通过利用 DataFlow 的模块化抽象以及基于图形的多智能体工作流引擎，弥合了高层人类意图与底层数据处理执行之间的差距。基于 LangGraph [2]，该智能体层通过有状态执行图协调一组专用智能体，将自然语言指令转化为可执行、自修正且优化的数据准备流水线。

5.1 智能体角色

为了实现自主流水线构建与代码合成，系统将职责分解至一组专业智能体。每个智能体封装了特定逻辑，并与 DATAFLOW 核心组件进行交互：

- **意图分析智能体**：接收用户高层次的自然语言查询，并将其分解为一系列可执行的子意图，为流水线提供基础蓝图。
- **数据路由智能体**：分析提供的输入数据以确定任务类别进行路由，或在未提供数据时生成合成数据占位符，以支持无数据执行。
- **操作符检索智能体**：接收特定的子意图作为输入，并利用 RAG 从 DATAFLOW 库中检索最相关的现有操作符作为潜在候选。
- **操作符序列化智能体**：评估候选操作符的输入/输出兼容性，以选择最佳匹配，或在检测到功能缺口时输出新操作符的详细规格。
- **操作符合成智能体**：接收缺失函数的规范，利用 RAG 生成上下文感知的代码，并执行自动化单元测试，直至代码可执行。
- **操作符复用智能体**：评估生成的操作符代码质量，并创建可复用的 `prompt_template`，确保代码能够高效复用而无需重新编写。
- **流水线构建智能体**：协调将所有有效算子（包括现有和新合成的）组装成一个准备就绪的有向非循环图（DAG）结构以供处理。
- **流水线验证智能体**：在沙箱环境中执行组装好的流水线，以识别运行时错误，并自主调整连接或参数，输出一个经过验证的无错误流水线。
- **结果报告智能体**：综合工作流的详细信息和执行结果，生成一份全面的报告以及可执行的流水线产物作为最终解决方案。

5.2 智能流水线推荐

如图 6 所示，系统的核心能力通过构建在 DataFlow 框架之上的复杂智能体层得以实现。该层采用 LangGraph [2] 来协调基于图形的状态化工作流中的多个专用智能体。

意图分解 工作流始于系统接收用户的自然语言查询。意图分析智能体将这一高层次目标分解为一系列离散且可执行的子意图。同时，数据路由智能体评估输入数据集，以对任务进行分类并决定下游路由路径。若未提供数据集，该智能体将生成合成数据占位符，以支持完整的预演执行。

操作员合成 为了实现这些子意图，操作符检索智能体在 DATAFLOW 库中搜索相关操作符，操作符排序智能体对其兼容性进行评估。若发现功能缺口，操作符复用智能体首先判断是否可通过 `prompt_template` 复用现有代码来满足需求。仅当复用不可行时，操作符合成智能体才使用基于 RAG 的 `few-shot` 学习生成新代码。随后，代码将自动调试以确保稳定执行。

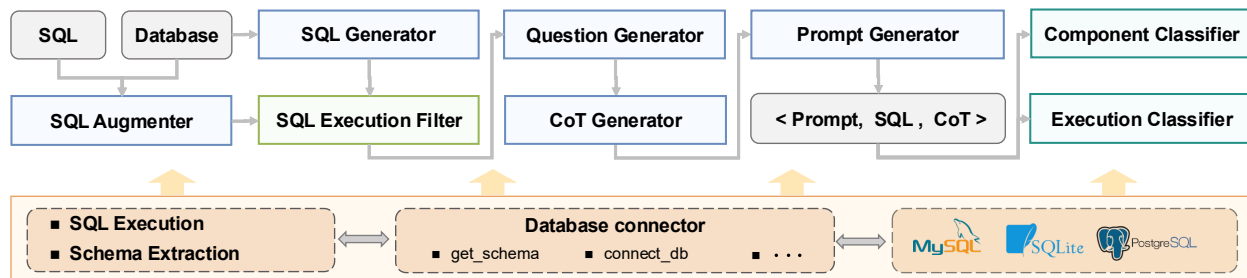


Figure 7 DATAFLOW 中 Text-to-SQL 流水线的整体框架

流水线装配 在所有检索或合成的算子都经过验证后，流水线构建智能体将它们组合成一个完整的流水线。它将流水线表示为有向无环图，并定义初始连接关系，以确保数据能够从源端流向目标端。

验证 系统随后运行集成测试。流水线验证智能体在沙箱环境中使用样本数据执行流水线，以检查连接性和运行时行为。如果出现错误，它会通过调整参数或连接来修复。流水线通过验证后，结果报告智能体生成报告，并输出最终的可执行流水线定义。

5.3 概要

总之，与 Data-Juicer 的 Agentic 方法 [6] 不同，后者主要受限于对预存算子静态库的参数化和序列化，DATAFLOW-AGENT 通过动态合成和调试缺失功能的可执行代码，实现了更程度的自主性。通过将“检索-复用-合成”策略与自校正验证环相结合，我们的系统超越了简单的配置生成，能够构建真正具备适应性的流水线，无需人工编码干预即可处理意外需求。

6 用例 & 流水线

DATAFLOW 集成了一系列丰富的数据流水线，覆盖了多种以文本为中心的任务领域，包括文本处理、数学推理数据、文本到 SQL 生成以及 Agentic 数据准备。此外，DATAFLOW 支持从 PDF 和教科书中进行结构化知识提取与规范化，从而支持模式构建、领域定位和指令合成等任务。

所有流水线均通过可重用的算子和声明式工作流规范实现，使用户能够以极少的工程投入灵活地组合、扩展和适应新场景。更详细的教程、流水线示例以及算子级文档可在网站上获取：<https://opendcai.github.io/DataFlow-Doc/>。

6.1 案例研究：DataFlow 中的文本到 SQL 数据流水线

我们首先设计了一组专门的、可重用的文本到 SQL 操作符，以确保模块化和可扩展性（参见第 6.1.1 节）。如图 7 所示，我们引入了两条流水线来构建高质量的文本到 SQL 数据集（参见第 6.1.2 节）。此外，第 6.1.3 节描述了 DATAFLOW 提供的数据库操作支持以及提示模板机制。

6.1.1 运算符

SQL 生成器。 SQL 生成器操作符利用数据库从头生成 SQL 查询，确保查询的多样性和有效性。定义了四个复杂度级别：简单、中等、复杂和高度复杂，并随机选择以指导大语言模型（LLM）通过清晰的

定义和 few-shot 例子生成不同难度的查询。数据库模式，包括所有关系表的 CREATE TABLE 语句以及随机采样的列值，为 LLM 提供了理解数据库所需的上下文。同时，还会随机提供高级 SQL 函数，以提高生成查询的真实性。由于自然语言问题通常需要查询特定条目，返回的列数也相应受到约束。在任务指令下，LLM 会生成有意义的 SQL 查询。在 DATAFLOW 框架中，只需替换相应的提示模板，该 SQL 生成器操作符即可自然地适应并复用于不同的数据库（例如，MySQL、SQLite、PostgreSQL）。

SQL 增强器 SQL 增强算子基于种子 SQL 生成多样且密切相关的增强型 SQL 查询，而非从零开始合成。我们提出了六种增强策略，从不同方向扩展 SQL 查询：(1) 数据值变换，(2) 查询结构修改，(3) 业务逻辑变更，(4) 复杂度提升，(5) 引入高级 SQL 特性，以及 (6) 性能与最优化。通过 few-shot 提示随机选择并应用这些类别。数据库模式和数值作为上下文信息提供。给定原始 SQL 查询和任务指令，该增强算子将生成其对应的增强 SQL 版本。

Text2SQL 一致性过滤器。 对于现有的自然语言问题与 SQL 查询配对，可能会出现两者不一致的情况，即内容不对应。此类有问题的数据需要被过滤掉，这可通过大语言模型实现，该模型会分析问题与 SQL 在内容上是否一致。

SQL 执行过滤。 并非所有生成的 SQL 查询都是有效或高效的。因此，SQL 执行过滤算子从两个方面对查询进行过滤：(1) 该 SQL 查询能否在目标数据库上成功执行；(2) 其运行时间是否超过预设阈值，若超过则予以丢弃，以确保系统的响应性。

问题生成器 问题生成器根据 SQL 生成语义等价的自然语言问题。自然语言问题分为以下风格类型：(1) 语气与正式程度：正式与非正式；(2) 句法结构与意图：祈使、疑问和陈述；(3) 信息密度与清晰度：简洁、描述性、模糊和隐喻性；(4) 交互模式：角色扮演和程序化。前两类涵盖具有明确用户意图的查询，而模糊和隐喻性风格则涉及不明确或比喻性语言。随机选择一种目标语言风格，并提供数据库模式作为上下文。基于任务指令和生成的 SQL 查询，大语言模型生成自然语言问题。

思维链生成器 思维链 (CoT) 推理通过将复杂任务分解为一系列更小、可管理的子问题，提升了模型解决复杂任务的能力。生成 CoT 推理迹需要任务指令、数据库模式、生成的自然语言问题以及生成的 SQL 查询。大语言模型会生成一个完整的推理链，涵盖中间推理步骤和最终的 SQL 查询。在 CoT 验证过程中，从推理链中提取生成的 SQL。只有当生成 SQL 在给定数据库上的执行结果与参考 SQL 一致时，该 CoT 过程才被视为有效解决方案。

提示生成器 作为模型的主要输入，提示 (prompt) 包含推理所需的必要信息。为了促进可靠的文本转 SQL 生成，一个结构良好的提示不仅应包含自然语言问题，还应包括数据库模式和具体的任务指令，以指导模型。提示生成操作符将这些组件合成最终的提示。

SQL 组件分类器。 对 SQL 查询进行分类可实现对其结构复杂性的深入分析。根据 Spider [70] 的评估标准，SQL 查询依据其语法组件的数量和复杂度被划分为四个难度等级：简单、中等、困难和超困难。这些语法组件包括列选择、SELECT 子句中聚合函数的使用，以及诸如 GROUP BY、ORDER BY、INTERSECT 或嵌套子查询等高级结构。SQL 组件分类器操作符根据既定标准将每个 SQL 查询分配到上述类别之一。

SQL 执行分类器。模型能否为给定的自然语言问题生成正确的 *SQL* 也是衡量难度的一个有意义指标。在 *SQL* 执行分类器操作符中，大模型被指令对同一输入提示重复生成 *SQL* 查询 k 次，并统计成功执行的次数，记为 n 。随后我们根据 $\frac{n}{k}$ 对难度等级进行分类。与 *SQL* 组件分类器操作符不同，执行难度具有模型依赖性：能力更强的大模型在同一任务上能获得更高的成功率，因此被认为具有更低的执行难度。

6.1.2 流水线

在 DATAFLOW 的设计理念中，流水线根据其功能被分解为独立的算子单元，从而实现算子的最大化复用。如图 7 所示，设计的算子被组合成两条流水线，以支持不同场景下的 *SQL* 数据合成。

SQL 生成流水线 该流水线基于数据库模式从零生成 *SQL*。首先，使用 *SQL* 生成器操作符生成初始 *SQL* 声明，随后通过 *SQL* 执行过滤器移除质量较低或不可执行的 *SQL*。接着，问题生成器为每条 *SQL* 查询生成对应的自然语言问题，思维链生成器操作符生成推理步骤 (CoT)，提示生成器构建提示内容。最后，*SQL* 组件分类器和 *SQL* 执行分类器为数据分配难度标签。

SQL 优化流水线 该流水线从现有的种子 *SQL* 开始生成数据。首先，流水线使用 *SQL* 执行过滤器验证种子 *SQL* 的质量，文本到 *SQL* 一致性过滤器则移除那些 *SQL* 与自然语言问题不一致的样本。随后，*SQL* 增强器基于种子 *SQL* 生成增强后的 *SQL*。后续步骤与 *SQL* 生成流水线相同：使用 *SQL* 执行过滤器过滤低质量 *SQL*，通过问题生成器生成自然语言问题，利用思维链生成器产生思维链推理，通过提示词生成器组合提示词，并最终使用 *SQL* 组件分类器和 *SQL* 执行分类器分配难度标签。

6.1.3 DataFlow 支持机制

数据库管理模块 在流水线中，高效可靠的交互机制作为核心基础设施，确保了工作流的稳定执行。为此，我们实现了 Database Manager 模块，该模块封装了数据库交互的底层细节，并提供统一、高效且可扩展的编程接口。Database Manager 在高并发工作负载下提升了处理吞吐量，并抽象了模式元数据的获取，从而降低了上层对潜在数据库结构的依赖。

为实现跨数据库兼容性，我们引入了抽象基类 DatabaseConnector。该类定义了一组标准化的接口，包括 connect_db (建立数据库连接)、execute_sql (执行 *SQL* 语句并返回结果) 以及 get_schema (获取完整的模式元数据)。针对每种数据库系统，开发者只需继承此基类并实现特定系统的驱动调用和错误处理逻辑，即可无缝集成到整个系统中。

提示模板模块 在生成 *SQL* 时，不同的场景 (如 CRUD 查询、向量搜索 *SQL*，或按不同难度规格分类的 *SQL*) 需要使用不同的提示模板。为了在这些不同需求下最大化操作符的可复用性，DATAFLOW 引入了提示模板模块。该设计使得 *SQL* 生成器操作符可以通过简单替换提示类，在不同场景中重复使用。实际应用中，用户只需在新的提示类中重新实现 build_prompt 方法，而无需修改 *SQL* 生成器操作符本身。

Table 2 预训练数据过滤：在通用基准测试中，使用 300 亿 token 规模的训练数据训练的模型性能对比。

| Methods | ARC-C | ARC-E | MMLU | HellaSwag | WinoGrande | Gaokao-MathQA | Avg |
|------------------------|-------|-------|-------|-----------|------------|---------------|--------------|
| Random-30B | 25.26 | 43.94 | 27.03 | 37.02 | 50.99 | 27.35 | 35.26 |
| Qurating-30B | 25.00 | 43.14 | 27.50 | 37.03 | 50.67 | 26.78 | 35.02 |
| FineWeb-Edu-30B | 26.45 | 45.41 | 27.41 | 38.06 | 50.43 | 25.64 | 35.57 |
| DataFlow-30B | 25.51 | 45.58 | 27.42 | 37.58 | 50.67 | 27.35 | 35.69 |

Table 3 SFT 数据过滤：在数学、代码和知识基准上，对不同 5k 数据集过滤方法的比较。

| Methods | Math | | | | | | Code | | | Knowledge | | |
|-----------------------------------|------|-------|--------|---------|----------|-------------|-----------|------|-------------|-----------|--------|-------------|
| | math | gsm8k | aime24 | minerva | olympiad | Avg | HumanEval | MBPP | Avg | MMLU | C-EVAL | Avg |
| Alpaca(random) | 54.9 | 77.2 | 13.3 | 14.0 | 27.0 | 37.3 | 71.3 | 75.9 | 73.6 | 71.8 | 80.0 | 75.9 |
| Alpaca(filtered) | 60.3 | 80.0 | 13.3 | 14.7 | 30.7 | 39.8 | 73.8 | 75.7 | 74.8 | 71.8 | 80.0 | 75.9 |
| WizardLM(random) | 61.1 | 84.2 | 6.7 | 18.0 | 29.3 | 39.9 | 75.6 | 82.0 | 78.8 | 71.8 | 79.2 | 75.5 |
| WizardLM(filtered) | 69.7 | 88.8 | 10.0 | 19.9 | 35.4 | 44.8 | 77.4 | 80.4 | 78.9 | 71.9 | 79.6 | 75.8 |
| DataFlow-SFT-15K(random) | 72.6 | 89.6 | 13.3 | 37.9 | 32.9 | 49.3 | 79.9 | 75.9 | 77.9 | 72.1 | 80.0 | 76.1 |
| DataFlow-SFT-15K(filtered) | 73.3 | 90.2 | 13.3 | 36.0 | 35.9 | 49.7 | 82.9 | 74.9 | 78.9 | 72.2 | 80.4 | 76.3 |

7 实验

在本节中，我们展示了涵盖文本、数学和代码数据准备的全面实验，以及使用 DATAFLOW 构建的 Text-to-SQL 和 AgenticRAG workflows。除了使用 Recall [9, 54] 框架进行训练的 AgenticRAG 情景外，其余所有实验均采用 LLaMA-Factory [74] 训练框架进行。我们进一步整合这些模态，以评估模型在多样化任务上的通用指令微调性能。

7.1 文本数据准备

7.1.1 实验情景

我们使用我们的 DATAFLOW 系统，评估高质量文本数据准备对预训练 (PT) 和监督微调 (SFT) 的影响。我们的实验涵盖了三种互补的场景：

(1) 预训练数据过滤 (300 亿规模)。从 SlimPajama-627B 语料库中，我们提取了一个包含 100B token 的子集，并应用了多个 DATAFLOW 文本预训练过滤器（在 `dataflow/operators/text_pt/filter` 中实现）。对于每个过滤器，选取前 30%（约 30B token）。我们使用 Megatron-DeepSpeed 框架，从零开始训练一个 Qwen2.5-0.5B 模型，共训练 30B token。我们对比了四种情景：

- **Random-30B**：一个随机的 300 亿 token 子集。
- **FineWeb-Edu-30B**：基于 FineWeb-Edu 的教育筛选 [50]。
- **Qurating-30B**：Qurating 使用阈值对 [64] 进行筛选：educational_value ≥ 7.5 , facts_and_trivia ≥ 4.0 , required_expertise ≥ 5.0 , writing_style ≥ 1.0 。
- **DataFlow-30B**：所有 DATAFLOW PT 筛选器的交集，选择前 30%。

(2) SFT 数据过滤 (5K 规模)。为了研究小规模 SFT 数据质量，我们使用 LLaMA-Factory 在 WizardLM 和 Alpaca 数据集上对 Qwen2.5-7B 基础模型进行微调。对于每个数据集，我们将随机抽取的 5K 个样

Table 4 对话合成：在不同 15K SFT 数据源下，Qwen2.5-7B 在对话领域数据集和通用基准上的性能对比。

| Model | Conversation Benchmarks | | | General Benchmarks | | | |
|----------------------------|-------------------------|-------|------|--------------------|------------|------------|--------------|
| | TopDial | Light | Avg | MMLU | AlpacaEval | Arena-Hard | Avg |
| Qwen2.5-7B | 7.71 | 7.79 | 7.75 | 71.45 | 7.05 | 0.60 | 26.36 |
| + ShareGPT-15K | 7.75 | 6.72 | 7.24 | 73.09 | 3.70 | 1.30 | 26.03 |
| + UltraChat-15K | 7.72 | 6.83 | 7.28 | 72.97 | 3.97 | 0.80 | 25.91 |
| + DataFlow-Chat-15K | 7.98 | 8.10 | 8.04 | 73.41 | 10.11 | 1.10 | 28.21 |

本与通过 DATAFLOW 的 SFT 流水线筛选出的 5K 个样本进行了对比。此外，我们利用 DATAFLOW 的 Condor 生成器和 Condor 精炼流水线合成一个 15k 大小的数据集 DATAFLOW-SFT-15K，随后经过 DATAFLOW 的 SFT 过滤流水线（不包含 Instagram 过滤器）。基准测试包括全面的数学、代码和知识评估套件。

(3) 对话领域合成 (15K 规模)。我们使用 DATAFLOW 的对话生成流水线合成 DATAFLOW-CHAT-15K，并在此数据集上微调 Qwen2.5-7B-Base。基准方法包括 ShareGPT-15K、UltraChat-15K 以及它们的完整（非截断）版本。我们在领域特定任务（TopDial, Light）和通用基准（MMLU [23], AlpacaEval [42], Arena-Hard [41]）上进行评估。

7.1.2 实验结果

预训练 首先，从表 2 可以看出，在六个通用基准（ARC-C/E、MMLU、HellaSwag、WinoGrande、Gaokao-MathQA）上，DATAFLOW 方法取得了最高的平均得分（35.69），优于 Random（35.26）、FineWeb-Edu（35.57）和 Qurating（35.02）。尽管使用了相同的 30B token 预算，DATAFLOW 的多过滤器交集策略生成了更干净且语义一致性更高的数据集，从而使得基于 0.5B 规模的 Qwen2.5 模型在从零开始训练时具备更好的泛化能力。

SFT 在表 3 中，我们随后使用 Alpaca、WizardLM 以及 DATAFLOW 生成的数据对 5K 规模的 SFT 数据过滤进行了评估。对于所有三种数据源，DATAFLOW 的过滤流水线在数学、代码和知识基准上均持续优于随机采样。同时，结果也表明，DATAFLOW 构建的 SFT 语料库本身强于 Alpaca 和 WizardLM：即使不经过过滤，DATAFLOW-SFT-15K 在数学上的平均得分（49.3）也高于经过过滤的 Alpaca（39.8）和 WizardLM（44.8），并且在代码和知识任务上仍保持竞争力。此外，DATAFLOW-SFT-15K 的随机采样与过滤版本之间的性能差距较小（49.3→49.7），进一步表明 DATAFLOW 生成的数据本身已更加干净且信息量更丰富，因此无需过于激进的过滤即可达到最佳性能。

对话 最后，从表 4 可以看出，DATAFLOW-CHAT-15K 将整体通用基准平均分从 26.36 提升至 28.21，并将 AlpacaEval 从 7.05 提升至 10.11，表现优于 ShareGPT 和 UltraChat。

这些发现表明，高质量的合成数据在与 DATAFLOW 的精炼和过滤堆栈结合使用时，能够超越常用的人工收集指令数据集。

7.2 数学推理数据准备

7.2.1 实验情景

我们基于 DATAFLOW 推理流水线构建了一个高质量的合成数学推理数据集，并针对大规模推理生成进行了相应调整。我们的目标是对比三种训练来源：(1) 从 Open-R1 [28] 随机选取的 10K 子集，(2) 从 Synthetic-1 [43] 随机选取的 10K 子集，以及 (3) 我们使用 DATAFLOW 构建的 10K 合成 DATAFLOW-REASONING-10K 数据集。

数据综合方法。数据生成过程遵循 DATAFLOW 推理流水线的核心结构，包括三个阶段：

- **问题合成**。我们采用 NuminaMath 数据集作为高质量的初始数据集，并利用 o4-mini 模型与 DATAFLOW 的数学问题合成算子将其扩展为一个多样化的候选问题池。
- **质量验证**。所有候选问题均通过 DATAFLOW 的 MathQ-Verify [53] 模块进行验证，该模块可检测错误、模糊或逻辑不一致的问题。低质量样本被移除，以确保正确性和鲁棒性。
- **思维链 (CoT) 生成**。对所有已验证的问题，我们使用 DATAFLOW 的思维链生成算子来提示 DeepSeek-R1 生成完整、逐步的推理迹。

与原始推理流水线相比，我们省略了种子级别的预验证阶段，因为 NuminaMath 本身就是一个经过筛选和验证的数据集。这在保持整体数据可靠性的同时，降低了计算开销。

我们评估了在不同 10k 合成数据集上微调的 Qwen2.5-32B-Instruct 在八个数学基准测试中的表现，包括 GSM8K [11]、MATH [24]、AMC23、奥数、Gaokao24-Mix、Minerva，以及 AIME 2024/2025。表 5 报告了完整结果。

生成超参数 对于非 AIME 问题，我们使用 `temperature = 0` 和 `top-p = 0.95`。对于 AIME 风格的问题，我们采用更具探索性的采样策略，设置 `temperature = 0.6`、`top-p = 0.95` 以及 `top-k = 20`。所有模型均在 10,000 个样本上使用 Qwen2.5-32B-Instruct 进行 1 轮次或 2 轮次的微调。

7.2.2 实验结果

我们的首个观察结果是，在 Synthetic-1 随机子集上进行训练仅带来了相对于基础模型的有限提升。尽管在 2 轮次后，AMC23 和 AIME 基准上出现了一些微小增益，但整体平均性能仍与仅使用指令的基准 (47.0 vs. 46.6) 相近。

相比之下，Open-R1 合成子集提供了更强的训练信号：经过两轮微调，平均得分从 48.7 提升至 54.2，表明 Open-R1 风格的思维链 (CoT) 数据对于增强 32B 模型的数学推理能力有效。在此基础上，我们的 DATAFLOW-合成数据集仅使用 10,000 个样本便实现了最强的整体提升，经过两轮微调达到最高平均性能 55.7，超越了 Open-R1 (54.2) 和 Synthetic-1 (54.0)。这些结果表明，结合经验证的 NuminaMath 种子、MathQ-Verify 过滤以及 DeepSeek-R1 驱动的思维链生成，能够提供更精确、多样且鲁棒的推理监督。

总体而言，实验表明，数据质量而非数据规模是影响数学推理性能的主导因子。即使在相同的 10k 大小下，我们的基于 DATAFLOW 的合成流水线始终优于现有的合成数据源。

Table 5 数学推理流水线：在不同合成数据训练情景下，Qwen2.5-32B-Instruct 的性能对比。

| Model | gsm8k | math | amc23 | olympiad | gaokao24_mix | minerva | AIME24@32 | AIME25@32 | Avg |
|------------------------------|-------|------|-------|----------|--------------|---------|-----------|-----------|-------------|
| Qwen2.5-32B-Instruct | 95.8 | 73.5 | 70.0 | 38.5 | 42.9 | 26.5 | 16.8 | 11.6 | 46.95 |
| <i>Trained with 1 epoch</i> | | | | | | | | | |
| + SYNTHETIC-1-10k | 92.9 | 71.8 | 52.5 | 38.4 | 23.1 | 24.3 | 35.6 | 34.0 | 46.6 |
| + Open-R1-10k | 91.5 | 72.3 | 65.0 | 38.4 | 20.9 | 24.6 | 43.0 | 33.5 | 48.7 |
| + DataFlow-Reasoning-10K | 93.9 | 72.3 | 72.5 | 38.7 | 38.5 | 26.5 | 35.9 | 34.5 | 51.6 |
| <i>Trained with 2 epochs</i> | | | | | | | | | |
| + SYNTHETIC-1-10k | 94.5 | 78.4 | 75.0 | 45.0 | 24.2 | 28.3 | 48.4 | 37.9 | 54.0 |
| + Open-R1-10k | 93.9 | 77.2 | 80.0 | 44.1 | 20.9 | 25.4 | 51.0 | 40.7 | 54.2 |
| + DataFlow-Reasoning-10K | 94.4 | 76.6 | 75.0 | 45.2 | 42.9 | 25.7 | 45.4 | 40.0 | 55.7 |

7.3 代码数据准备

7.3.1 实验情景

为了研究高质量代码指令数据对代码生成性能的影响，我们使用 Ling-Coder-SFT [12] 中的种子样本构建监督微调（SFT）数据集。我们首先从 Ling-Coder-SFT 语料库中随机抽取 20,000 个样本，并通过 DATAFLOW CodeGenDataset_APIPipeline 进行处理。这生成了三个不同规模的精炼代码指令数据集：DATAFLOW-CODE-1K、DATAFLOW-CODE-5K 和 DATAFLOW-CODE-10K，每个数据集均旨在为代码生成任务提供高质量、流水线优化的监督信号。

我们将合成数据集与两个广泛使用的基准进行比较，每个基准均子采样至 1000 个样本以保证公平性：

- **Code Alpaca (1k)**[5]：从 Code Alpaca 数据集随机抽取的样本。
- **Self-OSS-Instruct-SC2-Exec-Filter-50k(1k)** [63]：从 SC2-Exec-Filter 数据集随机选取的 1k 子集，该数据集包含基于执行的过滤。

模型在 DATAFLOW-CODE-1K、DATAFLOW-CODE-5K 和 DATAFLOW-CODE-10K 上使用全参数 SFT 进行微调。

随后，我们对两个基础模型进行实验：Qwen2.5-7B-Instruct 和 Qwen2.5-14B-Instruct。评估在四个代码基准上进行：(1) BigCodeBench [75]，(2) LiveCodeBench [30]，(3) CruxEval [22]，以及 (4) HumanEval [8]。最终性能报告为这四个基准上的平均值。表 6 中的所有数值均为百分比。

7.3.2 实验结果

表 6 显示，我们的合成数据集在所有基准上均持续提升 Qwen2.5-7B-Instruct 与 Qwen2.5-14B-Instruct 的代码生成性能。对于 7B 模型而言，即使仅使用 1k 的合成数据，其表现也已超越 Code Alpaca 与 SC2 执行过滤后的基准。具体而言，DATAFLOW-CODE-1K 在 BigCodeBench、LiveCodeBench 以及 CruxEval 上均优于原始模型，同时在 HumanEval+ 上仍保持竞争力。将监督规模扩展至 5k 与 10k 进一步提升了整体性能。特别是 DATAFLOW-CODE-10K 情景在所有指标上均取得最佳结果，包括 BigCodeBench 得分为 36.8，CruxEval(Input) 为 48.8，CruxEval(Output) 为 45.4，且整体平均得分高达 46.2，超过了相同数据规模下的 Code Alpaca-1K 与 SC2-Exec-Filter 基准。

对于更大的 Qwen2.5-14B-Instruct 模型，这些优势更加显著。尽管 Code Alpaca-1k 和 SC2 筛选对

Table 6 代码流水线：在不同 SFT 数据集设置下，Qwen2.5-7B-Instruct 与 Qwen2.5-14B-Instruct 的性能对比（所有数值单位为%）。

| Training Data | BigCodeBench | LiveCodeBench(v6) | CruxEval (Input) | CruxEval (Output) | HumanEval+ | Avg |
|--|--------------|-------------------|------------------|-------------------|-------------|-------------|
| <i>Trained on Qwen2.5-7B-Instruct</i> | | | | | | |
| Qwen2.5-7B-Instruct | 35.3 | 23.4 | 44.8 | 43.9 | 72.6 | 44.0 |
| + Code Alpaca-1K | 33.3 | 18.7 | 45.6 | 46.4 | 66.5 | 42.1 |
| + Self-OSS | 31.9 | 21.4 | 46.9 | 45.9 | 70.1 | 43.2 |
| + DataFlow-Code-1K | 35.5 | 25.7 | 48.0 | 45.1 | 72.6 | 45.4 |
| + DataFlow-Code-5K | 36.2 | 26.4 | 48.6 | 45.0 | 73.2 | 45.9 |
| + DataFlow-Code-10K | 36.8 | 26.0 | 48.8 | 45.4 | 73.8 | 46.2 |
| <i>Trained on Qwen2.5-14B-Instruct</i> | | | | | | |
| Qwen2.5-14B-Instruct | 37.5 | 33.4 | 48.0 | 48.5 | 74.4 | 48.4 |
| + Code Alpaca-1K | 37.0 | 28.2 | 50.2 | 49.6 | 71.3 | 47.3 |
| + Self-OSS | 36.9 | 22.3 | 52.6 | 50.1 | 68.3 | 46.0 |
| + DataFlow-Code-1K | 41.4 | 33.7 | 51.0 | 50.9 | 77.3 | 50.9 |
| + DataFlow-Code-5K | 41.1 | 33.2 | 52.5 | 50.6 | 76.2 | 50.7 |
| + DataFlow-Code-10K | 41.9 | 33.2 | 52.9 | 51.0 | 76.2 | 51.0 |

原始 14B 模型带来了适度的提升，但我们的数据集在所有指标上均持续带来更强的增益。特别是，DATAFLOW-CODE-10K 的平均得分为 51.0，在 BigCodeBench 上达到 41.9，在 CruxEval(Input) 上为 52.9，在 CruxEval(Output) 上为 51.0。值得注意的是，强调可执行正确性的 LiveCodeBench 得分从 Code Alpaca-1k 的 21.9 提升至我们合成监督下的 33.2。这些结果表明，DATAFLOW 生成的数据提供了比现有开源来源更明确的执行基准信号和结构化推理提示。

总体而言，实验表明，即使在相同的样本规模下，DATAFLOW-驱动的合成方法也始终优于现有的开源代码指令数据集。从 1k 到 10k 样本的持续提升表明了一个简单趋势：随着高质量 DATAFLOW 训练样本的增加，模型在代码推理任务上的表现持续提升。

7.4 文本到 SQL 的数据准备

7.4.1 实验情景

为了评估 Text-to-SQL 数据生成的有效性，我们构建了一个包含 89,544 个高质量 Text-to-SQL 实例的训练语料库，称为 DATAFLOW-TEXT2SQL-90K。DATAFLOW-TEXT2SQL-90K 中的每个实例均包含自然语言问题、对应的 SQL 查询以及思维链推理迹。具体而言，这些数据通过种子 SQL 查询的系统性增强获得：37,517 个实例源自 Spider-train [70] 数据集，37,536 个来自 BIRD-train [40] 数据集，14,491 个来自 EHRSQL-train [36] 数据集。DATAFLOW 流水线确保了 SQL 结构、问题表述以及多步推理过程在语法和语义上的丰富多样性。

对于我们的方法（表格 7 中的 DATAFLOW-Text2SQL 行），模型仅在我们合成的语料库上进行微调，除非另有说明。在评估时，我们采用六个广泛认可的 Text-to-SQL 基准：Spider [70]、BIRD [40]、EHRSQL [36]、Spider-DK [18]、Spider-Syn [17] 和 Spider-Realistic [14]。在使用大模型进行推理时，我们研究了两种解码策略：贪婪解码（记为 Gre），其温度设为 0 以生成确定性输出；以及绝对多数投票（记为 Maj）。绝对多数投票策略在温度 0.8 下对每个输入样本采样 8 个候选响应，执行所有有效的 SQL 查询，并选择在候选结果中出现频率最高的执行结果所对应的查询作为最终预测。此外，我们还随机抽取了 50K 个样本以构建 DATAFLOW-TEXT2SQL-50K。为了对比，我们也从 SynSQL [37] 中随机抽取了相同数量的样本。

Table 7 文本到 SQL 流水线：大模型在主流基准上的表现。前两个块列出了闭源和开源的基础模型。后两个块展示了微调后的模型，第一列表示训练数据情景。

| LLM / Training Data | Spider dev | | Spider test | | BIRD dev | | EHRSQL | | Spider-DK | | Spider-Syn | | Spider-Realistic | | Average | |
|--|------------|------|-------------|------|----------|------|--------|------|-----------|------|------------|------|------------------|------|---------|------|
| | Gre | Maj | Gre | Maj | Gre | Maj | Gre | Maj | Gre | Maj | Gre | Maj | Gre | Maj | Gre | Maj |
| <i>Closed-source LLMs</i> | | | | | | | | | | | | | | | | |
| GPT-4o-mini | 70.4 | 71.0 | 82.4 | 83.7 | 58.8 | 61.5 | 37.9 | 43.1 | 73.3 | 74.4 | 60.5 | 61.6 | 64.4 | 66.7 | 64.0 | 66.0 |
| GPT-4-Turbo | 72.4 | 72.2 | 83.4 | 84.2 | 62.0 | 63.6 | 43.1 | 44.8 | 72.3 | 72.1 | 62.9 | 63.5 | 67.5 | 68.3 | 66.2 | 67.0 |
| GPT-4o | 70.9 | 70.7 | 83.2 | 84.9 | 61.9 | 64.0 | 44.9 | 45.5 | 72.9 | 73.5 | 59.6 | 62.3 | 66.5 | 66.7 | 65.7 | 66.8 |
| <i>Open-source LLMs</i> | | | | | | | | | | | | | | | | |
| DeepSeek-Coder-7B-Instruct | 63.2 | 63.2 | 70.5 | 73.2 | 43.1 | 48.0 | 28.6 | 33.9 | 60.9 | 64.1 | 49.9 | 51.7 | 58.7 | 58.9 | 53.6 | 56.1 |
| Qwen2.5-Coder-7B-Instruct | 73.4 | 77.1 | 82.2 | 85.6 | 50.9 | 61.3 | 24.3 | 36.9 | 67.5 | 73.6 | 63.1 | 66.9 | 66.7 | 70.5 | 61.2 | 67.4 |
| Qwen2.5-7B-Instruct | 65.4 | 68.9 | 76.8 | 82.6 | 46.9 | 56.4 | 20.9 | 32.1 | 63.7 | 71.8 | 54.2 | 60.0 | 56.7 | 63.6 | 54.9 | 62.2 |
| OpenCoder-8B-Instruct | 59.5 | 59.5 | 68.3 | 70.1 | 37.5 | 45.3 | 21.9 | 29.9 | 62.6 | 64.7 | 46.0 | 46.1 | 49.0 | 49.4 | 49.3 | 52.1 |
| Meta-Llama-3.1-8B-Instruct | 61.8 | 67.7 | 72.2 | 78.5 | 42.0 | 53.1 | 24.6 | 33.7 | 62.6 | 69.9 | 53.1 | 59.3 | 57.5 | 61.0 | 53.4 | 60.5 |
| Granite-8B-Code-Instruct | 58.5 | 59.2 | 64.9 | 68.6 | 27.6 | 32.5 | 16.0 | 22.6 | 50.7 | 54.4 | 45.0 | 46.8 | 48.8 | 49.4 | 44.5 | 47.6 |
| Granite-3.1-8B-Instruct | 58.3 | 65.0 | 69.8 | 75.3 | 36.0 | 47.2 | 19.6 | 32.3 | 60.0 | 66.5 | 47.7 | 53.8 | 46.5 | 57.1 | 48.3 | 56.7 |
| <i>Trained on Meta-Llama-3.1-8B-Instruct</i> | | | | | | | | | | | | | | | | |
| SynSQL(50K) | 67.1 | 73.9 | 72.7 | 78.6 | 49.1 | 55.2 | 33.6 | 40.8 | 63.8 | 66.1 | 59.6 | 63.5 | 69.3 | 71.6 | 59.3 | 64.2 |
| SynSQL(90K) | 68.2 | 74.6 | 73.4 | 78.5 | 51.1 | 54.9 | 31.8 | 38.0 | 61.8 | 67.4 | 58.9 | 63.6 | 69.0 | 70.9 | 59.2 | 64.0 |
| SynSQL(2.5M) | 70.6 | 73.7 | 78.3 | 82.5 | 58.9 | 62.0 | 35.1 | 37.0 | 72.3 | 74.7 | 61.0 | 63.1 | 67.9 | 69.4 | 63.4 | 66.1 |
| Spider+BIRD+DataFlow-Text2SQL-90K | 74.9 | 79.2 | 78.4 | 82.3 | 53.4 | 58.9 | 28.4 | 36.5 | 67.7 | 69.7 | 66.6 | 69.1 | 74.4 | 75.0 | 63.4 | 67.2 |
| DataFlow-Text2SQL-50K | 69.9 | 76.8 | 75.1 | 80.1 | 51.4 | 57.6 | 28.0 | 36.4 | 65.9 | 68.1 | 61.3 | 67.5 | 69.6 | 73.5 | 60.2 | 65.7 |
| DataFlow-Text2SQL-90K | 71.4 | 76.4 | 75.8 | 80.0 | 54.6 | 56.8 | 55.5 | 56.3 | 66.5 | 67.7 | 61.6 | 67.3 | 71.4 | 72.7 | 65.3 | 68.2 |
| <i>Trained on Qwen2.5-Coder-7B-Instruct</i> | | | | | | | | | | | | | | | | |
| SynSQL(50K) | 77.1 | 82.1 | 81.8 | 84.8 | 54.0 | 59.3 | 33.1 | 44.1 | 67.1 | 69.5 | 68.0 | 70.6 | 77.2 | 80.3 | 65.5 | 70.1 |
| SynSQL(90K) | 79.2 | 83.1 | 82.3 | 84.4 | 56.2 | 59.4 | 31.4 | 41.4 | 65.0 | 70.7 | 67.2 | 70.7 | 77.0 | 79.9 | 65.5 | 69.9 |
| SynSQL(2.5M) | 81.2 | 81.6 | 87.9 | 88.3 | 63.9 | 66.1 | 34.9 | 40.0 | 76.1 | 77.8 | 69.7 | 69.6 | 76.2 | 78.0 | 70.0 | 71.6 |
| Spider+BIRD+DataFlow-Text2SQL-90K | 85.5 | 87.5 | 87.5 | 88.5 | 58.3 | 64.0 | 27.9 | 39.8 | 71.0 | 73.1 | 75.0 | 76.2 | 82.3 | 83.7 | 69.6 | 73.3 |
| DataFlow-Text2SQL-50K | 80.9 | 84.9 | 84.6 | 85.8 | 57.9 | 62.5 | 27.8 | 39.4 | 69.7 | 71.2 | 70.0 | 74.0 | 77.8 | 82.1 | 67.0 | 71.4 |
| DataFlow-Text2SQL-90K | 82.0 | 85.0 | 84.8 | 86.0 | 59.2 | 61.5 | 56.1 | 58.7 | 69.7 | 71.0 | 69.9 | 74.4 | 79.5 | 81.7 | 71.6 | 74.0 |

7.4.2 实验结果

如表 7 所示，生成的数据在多个主流基准上均带来了持续的性能提升，证明了 DATAFLOW [4] 的有效性。对于两种模型——Meta-Llama-3.1-8B-Instruct [21] 和 Qwen2.5-Coder-7B-Instruct [29]——在我们的生成数据上进行训练，其性能显著优于各自的基准以及其他竞争模型。当在生成数据上进行微调时，Qwen2.5-Coder-7B-Instruct 取得了显著提升：在 Spider-dev 上的执行准确率 (Gre) 从 73.4 提升至 82.0 (+8.6)，在 BIRD-dev 上从 50.9 提升至 59.2 (+8.3)，在具有挑战性的 EHRSQL 基准上从 24.3 提升至 56.1 (+31.8)。这些结果证实了 DATAFLOW-TEXT2SQL-90K 具有高质量和强大的训练实用性。

与其他训练数据集相比，我们的数据也展现出明显的优越性。在相近的数据规模下，基于 DATAFLOW-TEXT2SQL-90K 和 DATAFLOW-TEXT2SQL-50K 训练的模型始终优于基于 SynSQL [37]（分别为 SynSQL(90K) 和 SynSQL(50K)）训练的模型。具体而言，在 Spider-test 和 BIRD-dev 数据集上，基于 DATAFLOW-TEXT2SQL-50K 训练的模型分别达到了 84.6 和 57.9 的执行准确率 (Gre)，超越了 SynSQL(50K) [37] 所获得的 81.8 和 54.0。同样，基于 DATAFLOW-TEXT2SQL-90K 训练的模型不仅超越了基准模型，还优于 SynSQL(90K) [37]。值得注意的是，即使在远小得多的数据集上进行训练，经过 DATAFLOW-TEXT2SQL-90K 微调的模型在多个具有挑战性的基准测试中仍能达到与 SynSQL-2.5M [37] 相当的性能。这些提升凸显了 DATAFLOW 生成的训练数据更高的质量。

7.5 AgenticRAG 数据准备

Table 8 AgenticRAG 流水线：合成数据集与现有人工构建数据集的性能对比。所有数值均为确切匹配 (%)。“OOD-Avg”排除了每个训练数据集的领域内测试集。“DF-OOD (匹配)”在相同的领域内排除条件下，提供了 DF-AgenticRAG 的域外得分，确保比较的公平性。

| Training Data | HotpotQA | 2Wiki | Musique | Bamboogle | Avg | OOD-Avg | DF-OOD (matched) |
|---|----------|-------|---------|-----------|------|---------|------------------|
| Qwen-2.5-7B-Instruct | 25.0 | 25.8 | 9.9 | 27.2 | 22.0 | – | – |
| <i>Trained on HotpotQA (in-domain = HotpotQA)</i> | | | | | | | |
| HotpotQA-10k (1 epoch) | 40.2 | 41.9 | 16.7 | 42.4 | 35.3 | 33.7 | 33.8 |
| HotpotQA-10k (2 epochs) | 43.4 | 44.9 | 18.9 | 41.6 | 37.2 | 35.1 | 35.9 |
| HotpotQA-10k (3 epochs) | 45.3 | 48.0 | 20.3 | 40.8 | 38.6 | 36.4 | 37.4 |
| <i>Trained on Musique (in-domain = Musique)</i> | | | | | | | |
| Musique-20k (1 epoch) | 41.1 | 44.7 | 19.2 | 41.6 | 36.6 | 42.4 | 43.6 |
| <i>Trained on 2Wiki (in-domain = 2Wiki)</i> | | | | | | | |
| 2Wiki-30k (2 epochs) | 41.3 | 55.1 | 17.8 | 42.4 | 39.1 | 33.8 | 36.4 |
| <i>DF-AgenticRAG (raw results, for reference)</i> | | | | | | | |
| DataFlow-AgenticRAG-10k (1 epoch) | 39.3 | 42.6 | 17.3 | 41.6 | 34.3 | – | – |
| DataFlow-AgenticRAG-10k (2 epochs) | 43.1 | 44.6 | 19.9 | 43.2 | 37.7 | – | – |
| DataFlow-AgenticRAG-10k (3 epochs) | 42.6 | 45.5 | 20.2 | 46.4 | 38.7 | – | – |

7.5.1 实验情景

在 AgenticRAG 领域，多跳问题的自动生成长期以来一直是研究中的一个难题。本研究基于 **DataFlow AgenticRAG 流水线** 构建了一个规模为 10k 的多跳问题数据集，并与现有的主流多跳问答数据集（2Wiki-MultiHopQA [25]、Musique [58]、HotpotQA [68] 和 Bamboogle [51]）进行了对比分析。

数据集生成流水线的具体工作流如下：

- 从维基百科数据包中随机选取文档以形成初始文档集。为避免数据分布重叠对实验结果的干扰，已出现在测试基准中的文档被排除。
- o4-mini 模型结合 DATAFLOW AgenticRAG 的生成模块，用于根据筛选后的初始文档生成多跳问题的初始草稿。
- 验证模块用于筛选初始问题草稿的质量，剔除存在中间问题泄露、逻辑错误以及难度过高或过低的样本，最终形成一个高质量的多跳问题数据集，我们称之为 DATAFLOW-AGENTICRAG-10K。

本研究采用 ReCall [9] 框架完成模型训练与评估。在训练阶段，选择 Qwen2.5-7B-Instruct 作为基础模型，并使用 GRPO 强化学习算法进行模型最优化。在评估阶段，将模型的温度参数设置为 0.0。

对于检索组件，选择 E5-base-v2 [59] 作为检索器，2018 年的 Wikipedia 数据转储用作语料库。所有语料库索引和嵌入计算均通过 FlashRAG [32] 进行预处理。在整个训练和评估过程中，模型可自主指定检索的 topk 值，缺省的 topk 值设为 5，以在检索效率与性能之间取得平衡。

7.5.2 实验结果

表 8 报告了在四个多跳基准上的确切匹配性能。我们按训练数据集对结果进行分组，并通过移除每个数据集的领域内测试集（例如，HotpotQA 训练的模型不包含 HotpotQA）来计算分布外 (OOD) 平均值。为了公平地与我们的合成数据进行比较，我们还报告了 DF-OOD (匹配)，该方法对 DF-AgenticRAG-10k 也应用相同的领域内排除策略。

Table 9 知识提取：在不同推理和训练情景下，PubMedQA、Covert 和 PubHealth 上的准确率对比。

| Method (ACC) | PubMedQA | Covert | PubHealth |
|---------------------------------|---------------|---------------|---------------|
| CoT | 36.40% | 48.33% | 29.00% |
| RAG | 43.33% | 17.55% | 19.60% |
| SFT (DataFlow-Knowledge) | 53.40% | 68.33% | 40.86% |

与在 *HotpotQA* 上训练的模型进行比较。在 1 至 3 轮次中，HotpotQA-10k 的 OOD 平均得分分别为 33.7、35.1 和 36.4。在相同排除条件（不包含 HotpotQA）下，DF-AgentRAG 的得分分别为 33.8、35.9 和 37.4——尽管完全采用合成监督，仍持续保持与或超越 HotpotQA 的表现，提升幅度为 +0.1 至 +1.0 分。这表明 DF-AgentRAG 具备与广泛使用的手工构建数据集相当的泛化能力。

与 *Musique* 训练的模型进行比较。Musique-20k 在未使用 Musique 的情况下评估时，其域外平均得分为 42.4。在相同排除条件下，DF-AgentRAG（有效轮次规模为 2 20k）达到 43.6，比 Musique 高出 +1.2 分。这表明，我们的合成数据集不仅达到了与强人类标注的多跳基准相当的性能，而且在同一有效训练规模下表现更优。

与 *2Wiki* 训练模型的比较。2Wiki-30k 在 OOD 上的平均得分为 33.8。在相同排除条件（无 2Wiki）下，DF-AgentRAG（3 轮次，有效规模 = 3 万）达到 36.4，提升了 2.6 分，提升显著。这表明其与所有基准方法相比差距最大，凸显了我们合成问题强大的跨数据集泛化能力。

摘要。在所有训练方案和所有领域内排除的情况下，DF-AgentRAG-10k 在域外数据集中的表现始终为最佳或并列最佳，且在多个案例中（Musique、2Wiki）显著超越了人工构建的数据集。这些结果表明，我们的流水线能够生成具备卓越跨数据集泛化能力的多跳推理数据，说明高质量的合成数据不仅能够达到，而且持续超越现有人工标注多跳数据集的鲁棒性。

7.6 知识提取

7.6.1 实验情景

为了突破有限标注数据的限制，并充分利用互联网上的海量原始语料库，我们提出了知识提取流水线，这是一个用于语料库清洗和问答对合成的半自动化系统。该流水线使用 MinerU [46] 进行文本规范化，对长文档进行分段，过滤噪声或低质量句子，生成具有事实感知能力的问答对，并进行自动化质量检查，最终生成一个高质量的合成数据集，用于监督微调（SFT）。

在我们的实验中，训练数据源自从三个主要来源获取的 1.4 亿个 token 的原始医疗数据。第一个来源是 MedQA Books，该资源包含美国执业医师资格考试（USMLE）课程中广泛使用的 18 本医学教科书 [31]。第二个来源是来自 NCBI 图书架的 9,330 篇公开可获取的 StatPearls 文章 [66]。第三个来源包含从 16 家专业指南提供方聚合的 45,679 份临床指南文档 [10]。这些语料库作为知识提取流水线的输入，将其转换为结构化、高质量的问答数据集，记为 DATAFLOW-KNOWLEDGE，适用于模型训练。

在模型训练方面，我们基于 DATAFLOW 生成的数据集对 Qwen2.5-7B-Instruct 进行微调。SFT 过程共进行 37,500 步，覆盖五轮次。作为对比，我们还评估了一个 zero-shot 思维链（Chain-of-Thought, CoT）提示基准和一个基于检索增强生成（RAG）的基准，该基准采用 $\text{top-}k = 10$ 检索策略，并使用

`medcpt-query-encoder` 作为查询编码器，`medcpt-article-encoder` 作为文档编码器。所有基准在推理阶段采用相同的超参数设置。

我们在三个医学问答基准上评估我们的模型：PubMedQA [33]，专注于生物医学研究问题；Covert [45]，用于评估临床知识和推理能力；PubHealth [34]，针对公共卫生虚假信息分类。

7.6.2 实验结果

表 9 展示了在所有基准上的准确率结果。CoT 基线在各项任务中表现均不理想，表明仅依靠 zero-shot 推理对于医学问答系统而言是不够的，缺乏更针对性的监督无法取得良好效果。RAG 基线在 PubMedQA 上提供了适度的改进，但在 Covert 和 PubHealth 上仍不稳定，且显著低于预期表现，说明仅依赖检索无法替代针对结构化领域数据的显式训练。

相比之下，基于 DATAFLOW-KNOWLEDGE 合成数据训练的 SFT 模型在所有基准测试中均取得了最高准确率，显著超越了 CoT 提示和基于 RAG 的方法。值得注意的是，该模型在 PubMedQA 和 Covert 上的绝对准确率提升超过 15-20 个百分点，在 PubHealth 上也提升了 11 个百分点，这表明我们知识提取流水线生成的清洗和结构化问答对提供了更强大的监督信号。

总体而言，这些结果表明，通过针对特定的 DATAFLOW 流水线进行筛选和验证的高质量合成问答数据，能够显著提升通用模型在目标领域的推理能力，其表现优于推理时提示和检索增强的基准方法。

7.7 通过 DataFlow 实现统一的多领域数据准备

7.7.1 实验情景

数据构建 为了评估统一数据准备在跨模态特定推理任务中的效率和有效性，我们构建了一个整合的训练语料库，该语料库结合了数学、代码和通用指令数据。所有数据均通过 DATAFLOW 框架生成或筛选，具体如下：

- **数学**。我们使用 DATAFLOW 思维链流水线生成高质量的数学问题及其思维链（CoT）解法，以 MATH 数据集作为种子输入。我们随机采样 3k 个样本用于训练。
- **代码**。代码数据通过基于 20,000 个随机采样的 LingoCoder SFT 样本构建的 DATAFLOW CodeGenDataset_APIPipeline 生成。我们生成了 1,000 至 10,000 条高质量代码指令，并在 Code Alpaca 和 SC2-Exec-Filter 上进行基准测试。其中 2,000 个样本用于训练。
- **文本 / 通用指令**。对于自然语言任务，我们采用 DATAFLOW Condor 生成器 + 优化器流水线来生成高一致性指令-响应和对话对。输出结果进一步通过 SFT 质量过滤流水线处理。我们随机采样 5k 个样本。

所有模型均在合并的 DATAFLOW-INSTRUCT-10K 语料库上使用全参数 SFT 进行微调。评估涵盖：(1) 七个数学基准，(2) 四个代码基准，以及 (3) MMLU [23] 和 C-Eval [27]，用于通用知识和推理。

基准。我们还额外将 DATAFLOW-INSTRUCT-10K 与基于 **Infinity-Instruct** (Inf) [39] 语料库构建的基准方法进行了比较，该语料库是一个大规模通用指令语料库，在指令微调中被广泛使用。包含两个基准方法：

Table 10 DATAFLOW-INSTRUCT-10K 在数学基准测试中的表现：对 Qwen2-7B-Base 与 Qwen2.5-7B-Base 系列模型进行微调后的结果（确切匹配率%）。

| Model | MATH | GSM8K | AMC23 | AIME24 | Minerva | Gaokao | Olympiad | Math-Avg |
|-----------------------------------|------|-------|-------|--------|---------|--------|----------|----------|
| <i>Models based on Qwen2-7B</i> | | | | | | | | |
| Qwen2-7B-Base | 21.2 | 55.9 | 15.0 | 0.0 | 9.9 | 30.8 | 7.7 | 20.1 |
| + Inf-10K | 45.6 | 81.7 | 25.0 | 3.3 | 11.8 | 24.2 | 11.1 | 29.0 |
| + Inf-1M | 45.4 | 79.2 | 25.0 | 0.0 | 13.2 | 22.0 | 10.4 | 27.9 |
| + DataFlow-Instruct-10K | 54.0 | 83.0 | 27.5 | 0.0 | 16.5 | 25.3 | 20.3 | 32.4 |
| Qwen2-7B-Instruct | 53.9 | 86.2 | 22.5 | 3.3 | 17.6 | 35.2 | 19.6 | 34.0 |
| <i>Models based on Qwen2.5-7B</i> | | | | | | | | |
| Qwen2.5-7B-Base | 62.8 | 67.1 | 45.0 | 10.0 | 17.6 | 27.5 | 29.6 | 37.1 |
| + Inf-10K | 40.2 | 30.9 | 25.0 | 3.3 | 9.2 | 27.5 | 21.8 | 22.6 |
| + Inf-1M | 50.6 | 82.0 | 27.5 | 0.0 | 22.1 | 30.8 | 20.0 | 33.3 |
| + DataFlow-Instruct-10K | 73.8 | 88.2 | 47.5 | 16.7 | 30.9 | 31.9 | 37.6 | 46.7 |
| Qwen2.5-7B-Instruct | 75.1 | 92.4 | 47.5 | 10.0 | 34.9 | 48.4 | 40.6 | 49.8 |

Table 11 DATAFLOW-INSTRUCT-10K 在代码与知识基准上的表现：微调后的 Qwen2-7B-Base 与 Qwen2.5-7B-Base 模型。

| Model | HumanEval | MBPP | Code-Avg | MMLU | C-EVAL | Knowledge-Avg |
|-----------------------------------|-----------|------|----------|------|--------|---------------|
| <i>Models based on Qwen2-7B</i> | | | | | | |
| Qwen2-7B-Base | 66.5 | 66.1 | 66.3 | 69.6 | 82.8 | 76.2 |
| + Inf-10K | 64.0 | 71.7 | 67.8 | 69.3 | 83.0 | 76.2 |
| + Inf-1M | 65.9 | 70.4 | 68.2 | 69.5 | 83.0 | 76.2 |
| + DataFlow-Instruct-10K | 64.6 | 67.7 | 66.2 | 69.4 | 82.8 | 76.1 |
| Qwen2-7B-Instruct | 73.8 | 65.3 | 69.6 | 69.9 | 82.0 | 76.0 |
| <i>Models based on Qwen2.5-7B</i> | | | | | | |
| Qwen2.5-7B-Base | 78.7 | 74.3 | 76.5 | 71.9 | 80.0 | 76.0 |
| + Inf-10K | 77.4 | 77.8 | 77.6 | 71.8 | 79.9 | 75.8 |
| + Inf-1M | 78.0 | 78.0 | 78.0 | 72.2 | 79.4 | 75.8 |
| + DataFlow-Instruct-10K | 80.5 | 76.7 | 78.6 | 72.1 | 80.2 | 76.2 |
| Qwen2.5-7B-Instruct | 81.7 | 79.4 | 80.6 | 71.8 | 79.6 | 75.7 |

- **Inf-10K**：用于 SFT 的 Infinity-Instruct 的随机 10k 子集。
- **Inf-1M**：Infinity-Instruct 的一个随机 100 万子集。

与 Inf-10K/1M 对比使我们能够评估通过 DATAFLOW 生成的高质量、领域特定的合成数据（数学、代码、文本）是否比大规模通用指令数据提供更稳定和可靠的改进。

7.7.2 实验结果

在数学、代码和知识评估套件中，我们统一的多领域数据准备策略为 Qwen2.5-7B 和 Qwen2-7B 模型带来了持续且稳健的性能提升。所有表格中一个显著的模式是，DATAFLOW-INSTRUCT-10K 在所有非 Instruct 微调模型中几乎始终表现最佳，且在许多情况下，尽管使用的数据量少几个数量级，仍能将与 Instruct 模型之间的差距缩小至仅 2-4 分。

数学推理。如表 10 所示，经 DATAFLOW 处理的数学数据带来了最大且最稳定的提升。对于 Qwen2.5-7B-Base，基于我们合成的数学子集进行训练，整体得分从 37.1 提升至 46.7，提升了：

- 在所有非指令模型中表现最佳，明显优于 Inf-10K (22.6) 和 Inf-1M (33.3)；
- 仅比指令模型低 3.1 分 (49.8)，表明有针对性的高质量合成数据几乎可以达到昂贵的人类对齐指令微调的性能。

Qwen2-7B 也呈现出类似趋势：DATAFLOW-INSTRUCT-10K 的整体得分达到 32.4，优于 Inf-10K 和 Inf-1M，且接近 Instruct 模型 (34.04) 的水平。这些结果表明，DATAFLOW 数学合成生成的数据相比通用推理生成数据，能够带来更稳定且更有效的提升。

代码生成。如表 11 所示，DATAFLOW-INSTRUCT-10K 在所有非 Instruct 模型中始终展现出最佳的 Code-Overall 性能。对于 Qwen2.5-7B-Base，DATAFLOW-INSTRUCT-10K 将 Code-Overall 从 76.5 提升至 78.6，优于 Inf-10K (77.6) 和 Inf-1M (78.0)，且距离 Instruct 模型 (80.6) 仅差 2.0 分。对于 Qwen2-7B-Base，DATAFLOW-INSTRUCT-10K 再次达到或超越所有 Inf 基准模型。

这些结果表明，添加多领域合成数据不会损害代码能力（这是混合领域 SFT 中的常见问题），并且通常还能提升代码能力。这进一步支持了 DATAFLOW 领域均衡的合成语料库的鲁棒性。

通用知识与自然语言处理。如表 11 所示，我们的统一数据集同样保留了强大的通用知识和推理能力。在 MMLU 和 C-Eval 上，DF-Gen-10K：

- 与基础模型持平或略有提升
- 避免了 Inf-10K 和 Inf-1M 中经常出现的回归问题，
- 频繁仅次于 Instruct 模型，证实了 DATAFLOW 生成的文本数据即使没有人工指令微调，也能提供高质量的监督。

摘要。这些结果共同表明，通过 DATAFLOW 生成的高质量、领域专用的合成数据，在数学、代码和知识领域均展现出最强的非指令型性能。DATAFLOW-INSTRUCT-10K 始终优于通用的推理生成数据 (Inf-10K/Inf-1M)，并且经常接近指令模型本身的性能。这凸显了 DATAFLOW 统一的、流水线驱动的数据准备方法在构建多能力大模型方面的有效性，且无需依赖大规模人工编写的指令语料库。

7.8 Agentic 协调

7.8.1 实验情景

我们在真实的数据处理和流水线构建任务上评估了所提出的智能体编排框架。具体而言，我们选取了 6 个代表性流水线作为基准。对于每个流水线，我们手动构建了在 3 个难度等级下的自然语言任务描述，共生成 18 个用户查询，用于评估在不同描述粒度下自动编排能力的表现。难度等级定义如下：

- **简单**。描述是明确的，直接指定了所需算子（或关键算子）的功能和主要处理步骤。
- **中等**。描述较为粗略，仅提供一般的处理目标和关键约束，未明确列出完整的算子序列。

- **难题**。仅提供高层次的要求或最终目标，关于中间步骤的提示极少，要求系统推断出完整的处理流程和算子组合。

对于每个任务，用户会提供一个关于目标的自然语言描述，系统必须自动编排由多个算子组成的流水线以满足要求。

评价指标 为了定量评估编排质量，我们采用外部大语言模型作为自动评判者。评估者在两种不同情景下将生成的流水线与真实值进行比较：

- **文本规范对齐**。预测的图谱将根据文本规范进行评估，以验证流水线结构是否满足详细的任务要求。
- **代码实现一致性**。通过与参考的 Python 实现进行比较，评估流水线在算子使用和处理步骤上的逻辑等价性。

基于这些比较，我们报告了 **LLM-Judge 得分** ($s \in [0, 1]$)，该得分衡量在相应评估情景下，生成流水线与参考流水线之间操作符覆盖和执行顺序的一致性。

7.8.2 实验结果

Table 12 按评估模式和描述难度划分的智能体编排性能。

| Metric | Easy | Medium | Hard | Overall |
|---|------|--------|------|---------|
| Text spec evaluation (pipeline mode) | | | | |
| Avg. LLM-Judge | 0.92 | 0.86 | 0.60 | 0.80 |
| Code GT evaluation (code mode) | | | | |
| Avg. LLM-Judge | 0.60 | 0.59 | 0.23 | 0.49 |

表 12 报告了在文本规范（流水线）和代码真实值（代码）评估下，不同难度级别上的 LLM-Judge 得分。总体而言，当依据文本要求进行评判时，该框架表现良好（**0.80** 总体得分），但与参考实现匹配时得分明显较低（**0.49** 总体得分），反映出代码级等价性要求更为严格。随着描述变得不够明确，性能逐渐下降：在流水线模式下，得分从 **0.92/0.86**（简单/中等）降至 **0.60**（困难），而在代码模式下下降更为严重，在困难级别达到 **0.23**，表明描述不充分的查询往往导致生成与单一真实程序发散的、但仍合理的操作符组合。

8 结论

总之，DATAFLOW 通过提供首个统一的、由大语言模型驱动的数据准备框架，填补了以数据为中心的大语言模型生态系统中的关键空白。它通过模块化且用户友好的编程接口，缓解了该领域长期存在的挑战——例如数据准备算法在共享、复现和比较方面的困难。该框架集成了近 200 个操作符、80 多个提示模板，以及用于服务和存储的统一抽象，这些组件共同构成了涵盖主要大语言模型数据领域的六个高质量流水线。大量实验表明，这些流水线取得了强劲的表现，常常达到当前最优水平，证实了 DATAFLOW 在领域特定定制与系统级标准化之间实现了有效平衡。

在此基础之上，DATAFLOW-CLI 和 DATAFLOW-Agent 通过支持快速模板生成、基于自然语言的工作

流构建以及可扩展的模块开发，进一步增强了系统的可扩展性。这些组件共同奠定了一个可持续且具备互操作性的数据准备生态体系，能够随着日益复杂的以数据为中心的人工智能工作流不断演进。

展望未来,我们旨在沿着多模态维度扩展 DATAFLOW-Ecosystem,包括 DATAFLOW-TABLE、DATAFLOW-GRAPH 以及 DATAFLOW-MULTIMODAL，以支持更丰富的数据类型和工作流。我们还计划开发面向特定领域的变体，例如 DATAFLOW-AI4S 和 DATAFLOW-INDUSTRY，以适应大规模生产环境的需求。这些扩展将拓宽 DATAFLOW 的适用范围，并强化其作为未来大语言模型数据准备领域中基础性支撑——以及通用协议——的作用，推动科研、工程实践与社区驱动创新的发展。

References

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. [arXiv preprint arXiv:2303.08774](#), 2023.
- [2] LangChain AI. Langgraph. <https://github.com/langchain-ai/langgraph>, 2024.
- [3] Tianyi Bai, Hao Liang, Binwang Wan, Ling Yang, Bozhou Li, Yifan Wang, Bin Cui, Conghui He, Binhang Yuan, and Wentao Zhang. A survey of multimodal large language model from a data-centric perspective. [arXiv preprint arXiv:2405.16640](#), 2024.
- [4] Qifeng Cai, Hao Liang, Chang Xu, Tao Xie, Wentao Zhang, and Bin Cui. Text2sql-flow: A robust sql-aware data augmentation framework for text-to-sql. [arXiv preprint arXiv:2511.10192](#), 2025.
- [5] Sahil Chaudhary. Code alpaca: An instruction-following llama model for code generation. <https://github.com/sahil280114/codealpaca>, 2023.
- [6] Daoyuan Chen, Yilun Huang, Zhijian Ma, Hesun Chen, Xuchen Pan, Ce Ge, Dawei Gao, Yuexiang Xie, Zhaoyang Liu, Jinyang Gao, et al. Data-juicer: A one-stop data processing system for large language models. In [Companion of the 2024 International Conference on Management of Data](#), pages 120–134, 2024.
- [7] Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, et al. Alpapasus: Training a better alpaca with fewer data. [arXiv preprint arXiv:2307.08701](#), 2023.
- [8] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL <https://arxiv.org/abs/2107.03374>.
- [9] Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z Pan, Wen Zhang, Huajun Chen, et al. Learning to reason with search for llms via reinforcement learning. [arXiv preprint arXiv:2503.19470](#), 2025.
- [10] Zeming Chen, Alejandro Hernández Cano, Angelika Romanou, Antoine Bonnet, Kyle Matoba, Francesco Salvi, Matteo Pagliardini, Simin Fan, Andreas Köpf, Amirkeivan Mohtashami, Alexandre Sallinen, Alireza Sakhaeirad, Vinitra Swamy, Igor Krawczuk, Deniz Bayazit, Axel Marmet, Syrielle Montariol, Mary-Anne Hartley, Martin Jaggi, and Antoine Bosselut. Meditron-70b: Scaling medical pretraining for large language models, 2023. URL <https://arxiv.org/abs/2311.16079>.
- [11] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. [arXiv preprint arXiv:2110.14168](#), 2021.

- [12] Codefuse and Ling Team. Every sample matters: Leveraging mixture-of-experts and high-quality data for efficient and accurate code llm, 2025. URL <https://arxiv.org/abs/2503.17793>.
- [13] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
- [14] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. Structure-grounded pretraining for text-to-sql. *arXiv preprint arXiv:2010.12773*, 2020.
- [15] Qianlong Du, Chengqing Zong, and Jiajun Zhang. Mods: Model-oriented data selection for instruction tuning. *arXiv preprint arXiv:2311.15653*, 2023.
- [16] Erich Gamma. Design patterns: elements of reusable object-oriented software, 1995.
- [17] Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R Woodward, Jinxia Xie, and Pengsheng Huang. Towards robustness of text-to-sql models against synonym substitution. *arXiv preprint arXiv:2106.01065*, 2021.
- [18] Yujian Gan, Xinyun Chen, and Matthew Purver. Exploring underexplored limitations of cross-domain text-to-sql generalization. *arXiv preprint arXiv:2109.05157*, 2021.
- [19] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.
- [20] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, 2003.
- [21] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- [22] Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024.
- [23] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.
- [24] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.
- [25] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*, 2020.
- [26] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.
- [27] Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Yao Fu, et al. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *Advances in Neural Information Processing Systems*, 36:62991–63010, 2023.
- [28] Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL <https://github.com/huggingface/open-r1>.

- [29] Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. arXiv preprint arXiv:2409.12186, 2024.
- [30] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. arXiv preprint arXiv:2403.07974, 2024.
- [31] Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams, 2020. URL <https://arxiv.org/abs/2009.13081>.
- [32] Jiajie Jin, Yutao Zhu, Zhicheng Dou, Guanting Dong, Xinyu Yang, Chenghao Zhang, Tong Zhao, Zhao Yang, and Ji-Rong Wen. Flashrag: A modular toolkit for efficient retrieval-augmented generation research. In Companion Proceedings of the ACM on Web Conference 2025, pages 737–740, 2025.
- [33] Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. Pubmedqa: A dataset for biomedical research question answering. In Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP), pages 2567–2577, 2019.
- [34] Neema Kotonya and Francesca Toni. Explainable automated fact-checking for public health claims. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 7740–7754, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.623. URL <https://aclanthology.org/2020.emnlp-main.623/>.
- [35] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Proceedings of the 29th symposium on operating systems principles, pages 611–626, 2023.
- [36] Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. Ehrsql: A practical text-to-sql benchmark for electronic health records. Advances in Neural Information Processing Systems, 35:15589–15601, 2022.
- [37] Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, et al. Omnisql: Synthesizing high-quality text-to-sql data at scale. arXiv preprint arXiv:2503.02240, 2025.
- [38] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Guha, Sedrick Scott Keh, Kushal Arora, et al. Datacomp-lm: In search of the next generation of training sets for language models. Advances in Neural Information Processing Systems, 37:14200–14282, 2024.
- [39] Jijie Li, Li Du, Hanyu Zhao, Bo-wen Zhang, Liangdong Wang, Boyan Gao, Guang Liu, and Yonghua Lin. Infinity instruct: Scaling instruction selection and synthesis to enhance language models. arXiv preprint arXiv:2506.11116, 2025.
- [40] Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. Advances in Neural Information Processing Systems, 36, 2024.
- [41] Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E Gonzalez, and Ion

- Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. arXiv preprint arXiv:2406.11939, 2024.
- [42] Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. AlpacaEval: An automatic evaluator of instruction-following models. https://github.com/tatsu-lab/alpaca_eval, 5 2023.
- [43] Justus Mattern, Sami Jaghouar, Manveer Basra, Jannik Straube, Matthew Di Ferrante, Felix Gabriel, Jack Min Ong, Vincent Weisser, and Johannes Hagemann. Synthetic-1: Two million collaboratively generated reasoning traces from deepseek-r1, 2025. URL <https://www.primeintellect.ai/blog/synthetic-1-release>.
- [44] meta llama. Introducing Meta Llama 3: The most capable openly available LLM to date, 2024. URL <https://ai.meta.com/blog/meta-llama-3/>. Accessed: 2024-05-02.
- [45] Isabelle Mohr, Amelie Wüthrich, and Roman Klinger. Covert: A corpus of fact-checked biomedical covid-19 tweets, 2022. URL <https://arxiv.org/abs/2204.12164>.
- [46] Junbo Niu, Zheng Liu, Zhuangcheng Gu, Bin Wang, Linke Ouyang, Zhiyuan Zhao, Tao Chu, Tianyao He, Fan Wu, Qintong Zhang, Zhenjiang Jin, Guang Liang, Rui Zhang, Wenzheng Zhang, Yuan Qu, Zhifei Ren, Yuefeng Sun, Yuanhong Zheng, Dongsheng Ma, Zirui Tang, Boyu Niu, Ziyang Miao, Hejun Dong, Siyi Qian, Junyuan Zhang, Jingzhou Chen, Fangdong Wang, Xiaomeng Zhao, Liqun Wei, Wei Li, Shasha Wang, Ruiliang Xu, Yuanyuan Cao, Lu Chen, Qianqian Wu, Huaiyu Gu, Lindong Lu, Keming Wang, Dechen Lin, Guanlin Shen, Xuanhe Zhou, Linfeng Zhang, Yuhang Zang, Xiaoyi Dong, Jiaqi Wang, Bo Zhang, Lei Bai, Pei Chu, Weijia Li, Jiang Wu, Lijun Wu, Zhenxiang Li, Guangyu Wang, Zhongying Tu, Chao Xu, Kai Chen, Yu Qiao, Bowen Zhou, Dahua Lin, Wentao Zhang, and Conghui He. Mineru2.5: A decoupled vision-language model for efficient high-resolution document parsing, 2025. URL <https://arxiv.org/abs/2509.22186>.
- [47] NVIDIA Corporation. Curating custom datasets for LLM training with NVIDIA nemo curator. <https://developer.nvidia.com/blog/curating-custom-datasets-for-llm-training-with-nvidia-nemo-curator/>, 2024. Accessed: 2025-11-28.
- [48] Malte Ostendorff, Pedro Ortiz Suarez, Lucas Fonseca Lage, and Georg Rehm. Llm-datasets: An open framework for pretraining datasets of large language models. In *First conference on language modeling*, 2024.
- [49] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [50] Guilherme Penedo, Hynek Kydliček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.
- [51] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, 2023.
- [52] Matthew Rocklin et al. Dask: Parallel computation with blocked algorithms and task scheduling. In *SciPy*, pages 126–132, 2015.
- [53] Chengyu Shen, Zhen Hao Wong, Runming He, Hao Liang, Meiyi Qiang, Zimo Meng, Zhengyang Zhao, Bohan Zeng, Zhengzhou Zhu, Bin Cui, et al. Let’s verify math questions step by step. arXiv preprint arXiv:2505.13903, 2025.

- [54] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. arXiv preprint arXiv: 2409.19256, 2024.
- [55] Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari Morcos. Beyond neural scaling laws: beating power law scaling via data pruning. Advances in Neural Information Processing Systems, 35:19523–19536, 2022.
- [56] Gemini Team, R Anil, S Borgeaud, Y Wu, JB Alayrac, J Yu, R Soricut, J Schalkwyk, AM Dai, A Hauth, et al. Gemini: A family of highly capable multimodal models, 2024. arXiv preprint arXiv:2312.11805, 10, 2024.
- [57] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971, 2023.
- [58] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. Transactions of the Association for Computational Linguistics, 10:539–554, 2022.
- [59] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. arXiv preprint arXiv:2212.03533, 2022.
- [60] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers), pages 13484–13508, 2023.
- [61] Yudong Wang, Zixuan Fu, Jie Cai, Peijun Tang, Hongya Lyu, Yewei Fang, Zhi Zheng, Jie Zhou, Guoyang Zeng, Chaojun Xiao, et al. Ultra-fineweb: Efficient data filtering and verification for high-quality llm training data. arXiv preprint arXiv:2505.05427, 2025.
- [62] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35:24824–24837, 2022.
- [63] Yuxiang Wei, Federico Cassano, Jiawei Liu, Yifeng Ding, Naman Jain, Zachary Mueller, Harm de Vries, Leandro von Werra, Arjun Guha, and Lingming Zhang. Selfcodealign: Self-alignment for code generation. arXiv preprint arXiv:2410.24198, 2024.
- [64] Alexander Wettig, Aatmik Gupta, Saumya Malik, and Danqi Chen. Qurating: Selecting high-quality data for training language models. arXiv preprint arXiv:2402.09739, 2024.
- [65] Tom White. Hadoop: The definitive guide. ” O’Reilly Media, Inc.”, 2012.
- [66] Guangzhi Xiong, Qiao Jin, Zhiyong Lu, and Aidong Zhang. Benchmarking retrieval-augmented generation for medicine, 2024. URL <https://arxiv.org/abs/2402.13178>.
- [67] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. arXiv preprint arXiv:2505.09388, 2025.
- [68] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In Proceedings of the 2018 conference on empirical methods in natural language processing, pages 2369–2380, 2018.

- [69] Ping Yu, Jack Lanchantin, Tianlu Wang, Weizhe Yuan, Olga Golovneva, Ilia Kulikov, Sainbayar Sukhbaatar, Jason Weston, and Jing Xu. Cot-self-instruct: Building high-quality synthetic prompts for reasoning and non-reasoning tasks. arXiv preprint arXiv:2507.23751, 2025.
- [70] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. arXiv preprint arXiv:1809.08887, 2018.
- [71] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: a unified engine for big data processing. Commun. ACM, 59(11):56–65, October 2016. ISSN 0001-0782. doi: 10.1145/2934664. URL <https://doi.org/10.1145/2934664>.
- [72] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Guoyin Wang, et al. Instruction tuning for large language models: A survey. ACM Computing Surveys, 2023.
- [73] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. Advances in neural information processing systems, 37:62557–62583, 2024.
- [74] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyang Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. arXiv preprint arXiv:2403.13372, 2024.
- [75] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widayarsi, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. arXiv preprint arXiv:2406.15877, 2024.

Appendix

A 作者贡献

- 李浩亮: 项目负责人, 项目创始人; 算法负责人及论文撰写。
- 马晓晨: 项目负责人, 项目创始人; 系统负责人及稿件撰写。
- 周刘: 项目负责人, 项目创始人; DATAFLOW-AGENT 项目负责人及论文撰写。
- Wong Zhen Hao: 核心贡献者, 项目创始人; 负责设计和开发推理流水线、AI4S 流水线以及 AgenticRAG 流水线。
- 赵正阳: 核心贡献者, 项目创始人; 负责文本流水线的实验设计与实施。
- 孟子默: 核心贡献者, 项目创始人; 负责文本流水线的设计与实验的系统开发与支持。
- 何润明: 核心贡献者, 项目创始人; 负责开发推理流水线并开展数学推理实验。
- 沈成宇: 核心贡献者, 项目创始人; DATAFLOW 评估流水线和推理流水线。
- 蔡启峰: 核心贡献者; 负责设计并执行文本到 SQL 流水线的实验。
- 韩兆阳: 核心贡献者; 负责知识清洗流水线的设计与实验实施。
- 强美怡: 核心贡献者; 科学可视化、宣传领导与测试。
- 冯亚林: 核心贡献者; 设计了 DATAFLOW 评估和 PDF2Model 流水线。
- 白天一: 核心贡献者; 负责代码流水线的设计与实验执行。
- 潘泽威: Contributor; 设计并执行了 DATAFLOW-AGENT 的操作符编写工作流及实验。
- 郭子怡: Contributor; 设计并执行了用于 DATAFLOW-AGENT 的算子复用工作流。
- Yizhen Jiang: Contributor; 支持代码流水线的设计与实验。
- 邓景文: Contributor; 开发了 VQA 提取流水线和操作符。
- 游启杰: Contributor; 开发了 AgenticRAG 流水线并进行了实验。
- 赖培超: Contributor; 负责开发 DATAFLOW-WEBUI 的前端部分。
- 郭天宇: Contributor; 开发音频到文本的算子。
- 蔡志豪: Contributor; 修复了错误, 并应用 DATAFLOW 实现了在 BAAI LIC 挑战赛中排名第一的成绩。
- 冯恒毅: Contributor; DATAFLOW 测试。
- 胡瑞: Contributor; 负责执行 DATAFLOW-INSTRUCT-10K 实验。
- 余文凯: Contributor; 实现了多个算子。
- 牛俊波: Contributor; 支持 MinerU 集成到知识清洗流水线中。

- Zeng Bohan: *Contributor*; 支持框架设计, 并为文本到图像组件提供服务。
- 安瑞川: *Contributor*; 支持框架设计, 并提供 VQA 相关组件设计。
- Lu Ma: *Contributor*; 实现了多个算子。
- 黄吉豪: *Contributor*; 集成 LightRAG 服务。
- 郑耀威: *Contributor*; DATAFLOW 数据与 LLaMA-Factory 的集成。
- 何聪辉: 项目指导老师; 项目指导以及 MinerU 与 DATAFLOW 的集成。
- 唐林鹏: 项目指导老师; 项目指导。
- 崔彬: 项目指导老师; 项目指导。
- 魏安 E: 项目指导老师; 项目指导。
- 张文涛: 通讯作者, 项目指导老师; 论文撰写与项目指导。