arXiv:2602.05134v1 [cs.LG] 4 Feb 2026

SEMPIPES – 面向表格机器学习流水线的可优化语义数据算子

Olga Ovcharenko [1] **Matthias Boehm** [1] **Sebastian Schelter** [1]

摘要

真实世界中的表格数据机器学习依赖于复杂的数据准备流水线，以完成预测、数据集成、增强和调试。设计此类流水线需要丰富的领域知识和大量工程投入，因此引出了一个问题：大语言模型（LLM）如何通过代码合成来支持表格机器学习。我们提出 SEMPIPES，这是一种新颖的声明式编程模型，可将大语言模型驱动的语义数据算子集成到表格机器学习流水线中。语义算子以自然语言描述数据转换，同时将执行工作交由运行时系统处理。在训练期间，SEMPIPES 会根据数据特征、算子指令和流水线上下文合成定制的算子实现。这种设计借助进化搜索引导的大语言模型代码合成，实现流水线中数据操作的自动优化。我们在多种表格机器学习任务上评估了 SEMPIPES，结果表明，语义算子显著提升了专家设计和智能体生成的流水线的端到端预测性能，同时降低了流水线复杂度。我们使用 Python 实现了 SEMPIPES，并将其发布于
h
t
t
ps://github.com/deem-data/sempipes/tree/v1。

1. 引言

大语言模型（大语言模型）正在迅速重塑以数据为中心的软件开发，其驱动力来自它们在代码合成和程序理解方面日益增强的能力。近期方法涵盖从人工智能辅助编程环境（GitHub，2025年；Anysphere，2025年）和交互式代码智能体（Yang 等人，2024b；Anthropic，2025年）到能够合成新程序的自主系统（Chan 等人，2025年；Toledo 等人，2025年；Aygün 等人，2025年；Nam 等人，2025a；Fang 等人，2025年；Nam 等人，2025b）。尽管前景可期，人工智能辅助方法仍引发了关于安全性、可控性以及非预期行为的担忧（Nolan，2025年；Reddit，2023年），并且在某些情况下甚至可能降低开发者生产力（Becker 等人，2025年）。一个关键原因在于，基于聊天的交互只能对程序行为提供粗粒度且含糊的控制（Karpathy，2025年），而且所生成的代码难以可靠地优化（Khattab 等，2024；Lee 等人，2025年；Shi 等人，2024）。

表格机器学习流水线面临的差距。表格数据仍然是机器学习研究与实践的核心关注点（van Breugel & van der Schaar，2024；Hollmann 等人，2025；Qu 等，2025；Fey 等人，2025；Tschalzev 等人，2024）。在实际场景中，表格数据通常来自数据湖、数据仓库、日志文件或 REST API 等异构来源，并且必须通过表格机器学习流水线进行连接、清理、增强（Hollmann 等，2023b）和编码（Polyzotis 等人，2017）。这些流水线支持多种任务，包括预测、数据集成、调试和后处理（Polyzotis 等人，2019；Karláš 等人，2024；Cappuzzo 等人，2025），整合了来自众多库的复杂代码（Psallidas 等人，2022），且在生产环境中执行成本高昂（Xin 等人，2021）。因此，为表格机器学习开发数据流水线既繁琐又容易出错，需要大量领域专业知识和工程投入。行业调查一致指出，数据准备是数据科学实践中最耗时、最不受欢迎的环节（Anaconda，2022）。近期研究探索了面向预测任务的全自动端到端流水线生成（Toledo 等人，2025；Aygün 等人，2025；Nam 等人，2025a；Fang 等人，2025；Yang 等人，2025；Nam 等人，2025b；Küken 等人，2025），并在受限的、源自 Kaggle 的基准测试中取得了令人鼓舞的结果（Chan 等人，2025）。然而，这些方法仍有若干问题尚未解决，缺点详情见 A.1 节。常见问题包括初始生成的代码中有很大比例存在错误、无法运行（Trirat 等人，2025），容易发生数据泄漏，以及适用范围不明确，尤其是能否用于表格预测之外的任务，例如数据集成。若干近期系统既不公开生成的代码，也不公开代理轨迹，而且其设计与特定基准测试紧密绑定，限制了更广泛的适用性。

[1]BIFOLD 与柏林工业大学。通讯请联系：OlgaOvcharenko<ovcharenko@tu-berlin.de>、Sebastian Schelter <schelter@tu-berlin.de>。

声明式编程。声明式编程正成为一种颇具前景的范式，可通过受控、模块化的方式将基于大语言模型的操作集成到软件系统中。声明式程序不指定如何计算结果，而是使用自然语言指令描述要计算什么；运行时系统则负责确定执行策略并协调大语言模型调用。这种将意图与执行分离的做法，有助于缓解脆弱、临时拼凑且难以复用的代码等常见问题。重要的是，声明式方法使运行时能够利用自动优化机会，例如选择或合成专用算子实现。声明式方法已在语言模型编程和数据管理领域得到应用：（1）DSPy（Khattab 等，2024）支持以声明方式指定大语言模型模块，并将其编译为可执行程序，再针对数据和模型选择进行自动优化（Opsahl-Ong 等，2024；Agrawal 等，2025）。（2）在数据库领域，语义查询处理通过人工智能驱动的算子扩展 SQL，从而支持对多模态数据进行查询，并借助近似技术实现成本与质量之间的权衡（Patel 等，2025；Jo & Trummer，2024；Liu 等，2025；Dorbani 等，2025；谷歌，2025；Databricks，2025；Liskowski 等，2025）。

SEMPIPES：通过语义数据算子扩展表格机器学习流水线。声明式编程通过将要计算的内容与计算的执行方式分离，为将大语言模型集成到表格机器学习流水线提供了一种原则性方法。基于这一范式，我们推出 SEMPIPES，这是一种新颖的编程模型，通过声明式、由大语言模型驱动的语义数据算子扩展表格机器学习流水线。SEMPIPES 流水线以 Python 指定，并有选择地将以数据为中心的操作委托给具备上下文感知能力、可优化的语义算子（高层概览见图 1）。在训练期间，SEMPIPES 使用大语言模型合成语义算子的定制实现，合成过程以数据特征、自然语言指令和流水线上下文为依据（第 3 节）。因此，LLM 调用仅在训练期间发生，其数量随语义算子的数量增长，而与数据规模无关，从而使大语言模型的使用量可预测，并且推理时无需依赖大语言模型。SEMPIPES 的设计支持半自动化的流水线构建，同时保留了在交互式笔记本环境中迭代开发流水线的能力，并避免了基于聊天的界面和手动复制粘贴代码所带来的脆弱性。流水线定义完成后，可通过由进化搜索引导、基于大语言模型的代码合成，自动调优其算子实现，以提升端到端预测性能（第 4 节）。在此范围内，SEMPIPES 并非自动化机器学习系统，也不会生成端到端流水线；相反，它引入语义数据算子以改进数据操作，并可集成到用户编写或智能体生成的流水线中。

贡献。我们的贡献如下 ws:

• 我们提出了一种新颖的编程模型，通过引入可优化的语义数据算子来扩展表格机器学习流水线，并利用大语言模型合成适配数据和流水线的自定义算子代码（第 3 节）。

• 我们开发了一种语义数据算子优化器，通过在树结构进化搜索的引导下，利用基于大语言模型的变异来演进合成算子代码，从而提升流水线的预测性能（第 4 节）。

• 我们在一个新设计的基准测试上对 SEMPIPES 进行了实证评估。该基准测试涵盖五项表格机器学习任务，包含十九条由专家编写和智能体生成的流水线。我们发现，集成经过优化的语义数据算子能够持续提升流水线的预测性能，并且通常还能简化代码。此外，我们展示了合成算子代码在特征工程、缺失值填补和零样本特征提取方面可与自定义方法相媲美（第 5 节）。

• 我们用 Python 实现了 SEMPIPES，将其作为 skrub 库的扩展，并以开放许可证发布于 https://github.com/deem-data/sempipes/tree/v1。

2. 相关工作

表格机器学习流水线通常使用 pandas 数据框（The pandas develop- ment team，2020）来准备、整合和清理数据，然后将 scikit-learn（Pedregosa 等人，2011）中的所谓“估计器”（scikit-learn，2025）组合起来，以编码特征、训练型号并进行超参数搜索。此类流水线在实践中已得到广泛采用（Psallidas 等人，2022）；截至 2025年10月，根据 pypistats.org 的数据，scikit-learn 和 pandas 软件包的月下载量合计已超过 6 亿次。此外，scikit-learn 的估计器抽象已被其他热门机器学习库采用，包括 SparkML（Meng 等人，2016）和 TensorFlow Transform（Baylor 等人，2017），并构成了当前机器学习流水线研究的基础（Karlaš 等人，2024；Petersohn 等人，2020；Graf- berger 等人，2023）。尽管广受欢迎，目前的表格机器学习流水线生态系统仍存在若干重大局限。DataFrame 和数据操作在 scikit-learn 的估计器抽象中并非一等公民；该抽象旨在对单个行数固定的表格应用线性变换序列（skrub，2025b）。因此，筛选、连接、聚合以及处理多个输入表等常见操作均不受支持。集成外部库（例如用于自动化特征工程的库）仍然十分困难（Schäfer 等，2025）。

过去，数据操作通常被排除在端到端优化方法之外（Yu等人，2021年），因为它们不可微。一种有前景的方法是

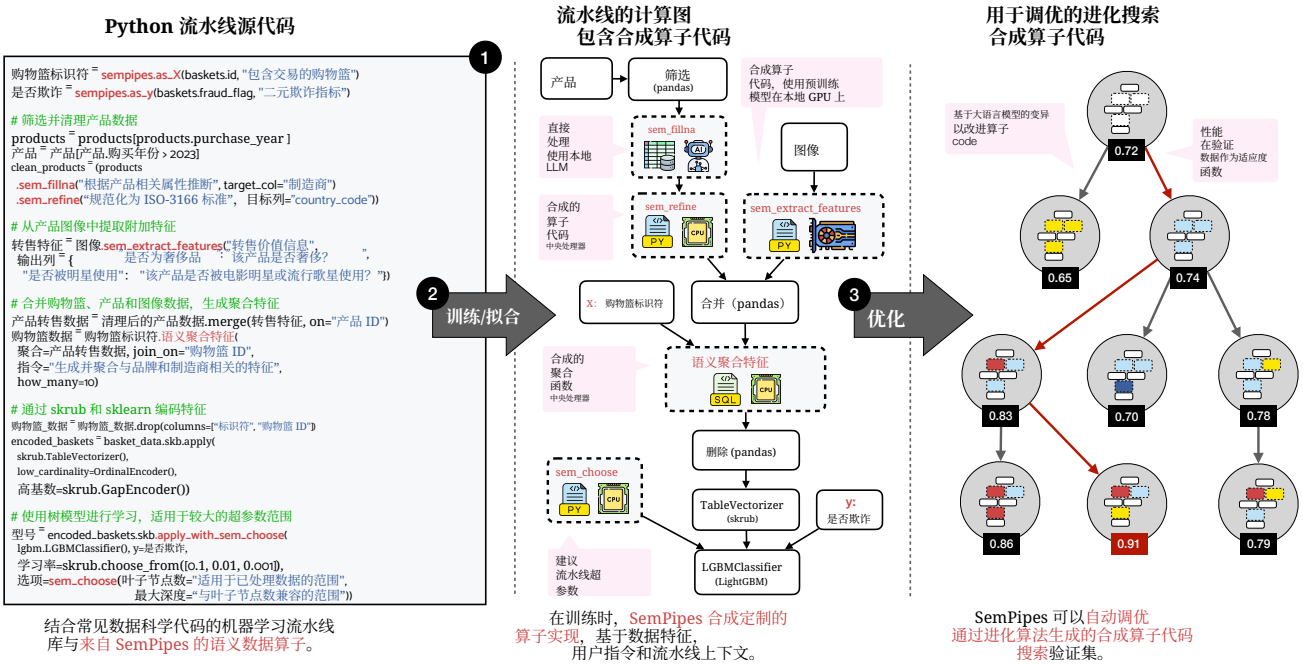


图 1：**S 的整体概览EMPIPES**： 机器学习流水线使用标准数据科学库在 Python 中定义，包括用于表格数据处理的 pandas、用于数值计算的 NumPy、用于特征编码的 scikit-learn，以及与 scikit-learn 生态系统兼容的模型。SEMPIPES 通过语义数据算子扩展这些流水线，这些算子将选定的以数据为中心的操作委托给大语言模型。 在训练期间，当流水线针对训练数据进行“拟合”时，SEMPIPES 会根据数据特征、自然语言指令和流水线上下文，为语义算子合成定制实现。合成代码可在本地中央处理器上执行，也可利用本地图形处理器上的预训练模型。 流水线定义完成后，可在验证集上调整其语义算子的实现。SEMPIPES 会依据给定的搜索策略（例如蒙特卡洛树搜索）执行进化搜索，并以验证性能作为适应度函数。在搜索过程中，算子实现会利用先前的性能得分和树结构记忆，通过反思式提示进行迭代变异。

克服这一局限的途径是提示优化（Yang 等人，2024a），即由代码生成大语言模型利用文本反馈和性能得分迭代改进程序（Hollmann 等人，2023b；Opsahl-Ong 等人，2024；Agrawal等人，2025年；Lee 等人，2025年；Aygün 等人，2025年）。机器学习流水线天然适合这一设定，因为验证性能提供了明确的优化目标函数。

我们将关于使用语义算子进行声明式编程以及智能体流水线生成的相关工作的更广泛讨论推迟到第 A.2 节。我们认为，SEMPIPES 为智能体提供了一种互补的抽象：可复用的基本构件，使其有可能将对高层流水线结构的推理与各个数据操作的合成和优化分离开来。

3. SEMPIPES 编程模型

SEMPIPES 提供了一种新颖的编程模型，可将大语言模型驱动的语义数据算子集成到表格机器学习流水线中，使其适应当前的数据和任务。这种抽象支持一种半自动化流水线构建方式，将选定的数据操作委托给大语言模型。SEMPIPES 面向跨多个表格数据集的复杂 Python 流水线，其中包含文本、图像或音频等多模态单元格；它还支持连接、聚合、清洗、预处理和编码，可通过 DataFrame 操作、scikit-learn 风格估计器、用户定义函数以及与 scikit-learn 生态系统兼容的模型来实现。

语义数据算子。我们为表格机器学习流水线引入了九种语义数据算子（SemOps）（见表 1），分为三类：(i) 四种用于特征生成、提取和数据增强的算子；(ii) 三种用于数据清洗与集成的算子；以及(iii) 两种用于流水线参数化的算子，包括语义列选择，以及为估计器提出超参数范围建议（Belkhiter等人，2025年）。每种算子都有明确定义的输入和输出类型，使用自然语言指令进行配置，并作用于数据框或估计器参数，可与skrub、pandas和scikit-learn无缝集成。图 1 展示了一个虚构欺诈检测用例的示例流水线。我们选择这些算子时，参考了启发式方法，

表 1：S 中的语义数据算子SEMPIPES。 此处，s 是一条自然语言指令， D、 D' 表示输入/输出数据框。右侧列显示其适用于数据框（DF）、多模态数据框（MM）和估计器（ET）的情况。

语义数据算子	定义	适用于 DF	MM	ET
sem_gen_特征$(D,\ k,\ s) \to D'$	在 D中生成 k 个特征列	✓		
sem_agg_特征$(D,\ D_{\text{agg}},\ c_{\text{join}},\ k,\ s) \to D'$	在 D_{agg}中生成 k 聚合特征列，在 c_{join}上与 D 连接	✓		
sem_提取_特征$(D,\ C_{\text{out}},\ s) \to D'$	从特征描述中提取特征列 C_{out} 和 D		✓	
sem_增强$(D, k, s) \to D'$	为 D 附加行	✓		
sem_填充缺失值$(D, c, s) \to D'$	填补 D的列 c 中的缺失值	✓	✓	
sem_清理$(D, c, s) \to D'$	清理 D的列 c	✓		
sem_细化$(D,\ c,\ s) \to D'$	重构 D 的列 c	✓		
sem_选择$(D,\ s) \to D'$	将 D 缩减为选定的列子集	✓	✓	✓
sem_选择$(E,\ s) \to H$	为估计器 E推荐超参数或配置选项			✓

例如skrub等开源库，以及近期强调将研究原型集成到生产代码中所面临挑战的研究（Schäfer等人，2025年）。我们的设计允许对哪些数据操作由大语言模型驱动的算子处理进行细粒度控制，并支持在 Jupyter 笔记本等交互式环境中开发。SEMPIPES 有意专注于以数据为中心的操作，而非模型训练；易于使用的模型已能很好地支持模型训练（Grinsztajn等人，2025年；Qu等人，2025年；Grinsztajn等人，2022年），而成熟的超参数调优方法也已广泛应用（Feurer与Hutter，2019）。

执行模型。SEMPIPES 利用 skrub DataOps 流水线（skrub，2025a），这是一种轻量级抽象，用于处理基于数据框的多表机器学习流水线（详细介绍见附录 A.3 ）。DataOps 将流水线表示为计算图，其中节点表示数据操作，边表示数据依赖。这种设计既支持惰性执行，也便于对流水线本身进行推理和重写。标准的 scikit-learn 流水线仅限于对单个表执行一系列线性变换，且表的行数固定。相比之下，DataOps 抽象将整个流水线（包括多表数据框操作）整合为统一的 scikit-learn 风格预测器，从而能够无缝处理跨多个表的复杂数据操作。为此，DataOps 流水线提供 fit 方法，用于端到端训练，并提供 predict 方法，用于对未见数据进行推理。SEMPIPES 利用计算图将语义数据算子集成为 scikit-learn 风格估计器。

训练时的特定算子代码合成。SEMPIPES 会在训练时，根据输入数据特征（例如面积、列名和统计量）、自然语言指令以及流水线上下文（包括从源代码和计算图中推断出的学习任务和模型类型），合成定制的算子实现（参见图 2中的简单示例）。根据算子类型和用户指令，SEMPIPES 会合成不同的代码变体：对于特征生成，它会生成基于中央处理器的 DataFrame 和聚合代码；对于多模态特征提取，它会生成代码，以便在本地 GPU 上运行轻量级预训练模型（例如，以零样本模式运行 HuggingFace 模型）。训练时只需进行少量大语言模型调用，其数量与语义算子的数量呈线性关系；生成的算子代码可高效执行，且推理时无需调用大语言模型。以算子为中心的设计还支持对合成代码进行细粒度验证：除了通过解析进行语法检查外，SEMPIPES 还会在一份输入数据样本上执行每个算子，并验证算子特定的输出约束，例如结果 DataFrame 的预期形状。有关各算子的代码合成以及在数据处理时直接调用大语言模型的替代实现，请参见A.4 节。

4. 优化语义数据操作

为了改进表格机器学习流水线中的数据操作，数据科学家通常会选择一个目标操作（例如特征生成或缺失值填补），重写其代码，集成外部库，并在验证集上重新运行流水线，以评估性能提升。他们会反复迭代，直到流水线达到预期目标或时间耗尽。SEMPIPES 通过树结构进化搜索引导基于大语言模型的变异（代码修改），调整其语义数据算子的合成代码，从而提升预测性能，自动化这一工作流程。

语义流水线。我们将语义机器学习流水线视为语义算子 $o(\cdot;\phi)$ 的组合，这些算子具有状态 ϕ（其合成代码）；此外还包含预测算子 $f(\cdot;\theta)$，后者具有可学习参数 θ（例如预测模型、降维方法或集成模型）。从概念上讲，每个输入 $(x, y) \in \mathcal{D}$ 都会先经过语义算子处理，其输出随后被预测算子用于生成预测结果 $\hat{y} = f(o(x;\phi);\theta)$。整个流水线的性能由标量效用函数 $\mu(y, \hat{y})$ 衡量，例如在留出数据上的准确率。由于语义算子 o 不可微，无法通过基于梯度的方法联合优化流水线参数 ϕ 和 θ。我们采用双层优化策略（Sinha 等人，2017年），在以下步骤间交替：(*i*) 优

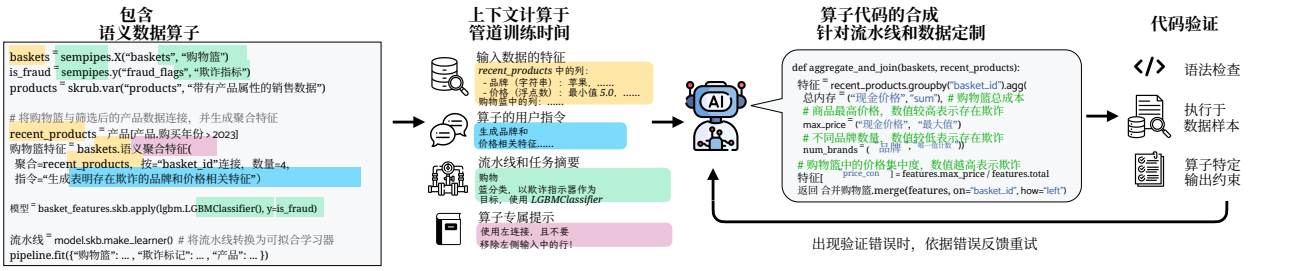


图 2：**语义数据算子的代码合成示例**。在训练时，SEMPIPES 会根据输入数据特征（例如面积、列名和统计量）、自然语言指令，以及学习任务和模型类型等流水线上下文，合成定制的算子代码。通过在一部分输入数据上执行合成代码，并检查算子特定输出约束，对其进行验证。

化固定语义算子状态下预测算子的可学习参数 θ，以及 (ii) 通过大语言模型引导的变异过程更新语义算子状态 ϕ。我们的目标是求解：

$$\underset{\phi}{\operatorname{argmax}}\ \mathbb{E}_{(x,y)\sim\mathcal{D}}\Big[\mu\big(y, f(o(x;\phi);\theta^*(\phi))\big)\Big], \qquad (1)$$

其中，内层优化针对给定的一组语义算子状态 ϕ，定义预测算子的最佳可学习参数 $\theta^*(\phi)$，即 $\theta^*(\phi) = \arg\max_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}}\big[\mu\big(y, f(o(x;\ \phi);\ \theta)\big)\big]$。对于公式（1）的优化，我们采用基于大语言模型的变异机制，根据性能反馈以及对输入数据、流水线上下文和过去语义算子状态的语义推理，提出改进后的语义算子状态。给定一些训练数据 $\mathcal{D}_{\text{train}}$，优化过程通过以下方式迭代交替进行内循环学习和外循环变异。在步骤 i 的内循环学习中，将当前语义算子状态固定为 $\phi^{(i)}$，并拟合流水线，以学习预测算子的参数，使其在 $\mathcal{D}_{\text{train}}$ 上的效用最大化，即 $\theta^{(i)} = \arg\max_\theta \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \mu\big(y, f(o(x;\ \phi^{(i)});\ \theta)\big)$。接下来，外循环变异首先评估 $\mathcal{D}_{\text{train}}$ 上的验证效用 $\sum_{(x,y)\in\mathcal{D}_{\text{train}}} \mu\big(y, f(o(x;\ \phi^{(i)});\ \theta^{(i)})\big)$（通常通过交叉验证），然后使用基于大语言模型的变异生成新的语义算子状态 $\phi^{(i+1)}$，以提高期望效用。

演化单个语义数据算子的代码。接下来，我们将详细说明如何在优化过程中迭代改进语义数据算子 o_j的语义算子状态 $\phi_j^{(i)}$。我们针对每种语义算子类型设计了函数 EVOLVE(o_j, $\mathbf{C}$, $\mathbf{V}^{(i)}$, $\mathbf{M}_j^{(i)}$, $\mathbf{X}_j$, $\mathbf{V}^{(i)}$; L) $\to \phi_j^{(i)}$, $m_j^{(i)}$ 该函数通过语言模型 L演化语义算子 o_j 的内部状态 $\phi_j^{(i+1)} \to \phi_j^{(i+1)}$（例如，其合成代码）。该函数输出演化后的状态 $\phi_j^{(i+1)}$ 以及相应的记忆轨迹 $m_j^{(i+1)}$，其中包含当前算子状态的尝试历史和推理过程。这些输出取决于流水线

算法1 使用进化树搜索和 LLM 引导的变异优化 SEMPIPES 流水线的语义算子状态 ϕ。

要求：包含语义数据算子的流水线 o、训练数据 $\mathcal{D}_{\text{train}}$、语言模型 L、搜索策略 π、效用函数 μ、预算 $B\phi^{(0)}$、$m^{(0)} \leftarrow$ 空状态和备忘录 $T \leftarrow$ 基于 $\phi^{(0)}$ 初始化搜索树，根节点为 $s^{(0)}$ 对于步骤 $i \in 0 \dots B$，如果 i > 0$s^{(i)} \leftarrow$ 根据策略选择下一个搜索节点 $\pi(T)\phi^{(i)}$，$m^{(i)} \leftarrow$ 演化 _ SEMOPS($\mathcal{D}_{\text{train}}$, $s^{(i)}$,π, L)$\tilde{\theta} \leftarrow$ 使用固定的 $\phi^{(i)}v^{(i)} \leftarrow$ 在 $\mathcal{D}_{\text{train}}$ 上拟合流水线 μ 使用 $\phi^{(i)}$、$\tilde{\theta}$ 对流水线进行效用评估 $\mathcal{D}_{\text{train}}$记录结果 $(v^{(i)}, \phi^{(i)}, m^{(i)})$，用于 $s^{(i)}$ 在 T中返回 ϕ 在 T中记录的最高效用函数 EVOLVE_SEMOPS($\mathcal{D}_{\text{train}}$, $s^{(i)}$, π, L)C $\leftarrow$总结流水线上下文 $\mathbf{V}^{(i-1)} \leftarrow [v^{(0)}, \dots, v^{(i-1)}]$ 到 $s^{(i)}$ 路径上的效用分数对于计算有向无环图 $\mathrm{I}_j \leftarrow$ 中按顺序排列的语义算子 o_j 从 T 中选择的启发状态 πM $_j^{(i-1)} \leftarrow$到 $s^{(i)}$ 路径上的备忘录 $[m_j^{(0)}, \dots, m_j^{(i-1)}]$ $\mathrm{X}_j \leftarrow$将流水线部分拟合至算子 o_j，使用 $\mathcal{D}_{\text{train}}\phi_j^{(i)}$，$m_j^{(i)} \leftarrow$ EVOLVE(o_j, C, $\mathrm{M}_j^{(i-1)}$, I_j, X_j,$\mathrm{V}^{(i-1)}$; L) 返回语义算子状态 $\phi^{(i)}$ 及其备忘录 $m^{(i)}$

上下文 $\mathbf{C}$（概述计算图的结构、任务类型和所用型号的类型），以及先前效用分数的序列 $\mathbf{V}^{(i)}$ 和记忆轨迹 $\mathbf{M}_j^{(i)}$，以及流水线中算子 o_j 的具体输入数据 $\mathbf{X}_j$。此外，该函数还可以接收外部“灵感” $\mathbf{I}_j$，例如在其他优化轨迹中表现良好的算子代码示例。更多详情请参阅 A.5 节。

用于流水线优化的树结构进化搜索。我们将迭代式双层优化过程与针对语义算子状态的树结构进化搜索相结合（Aygün 等人，2025年），如算法 1所示。搜索树中的每个节点 $s^{(i)}$ 都封装了该步骤所达到的效用值 T 以及 $v^{(i)}$ 生成相应语义算子时产生的记忆轨迹 $m^{(i)}$。

算子状态。搜索树的遍历与扩展由搜索策略控制 π，该策略会为每个优化步骤选择下一个起始节点。该策略可以通过蒙特卡洛树搜索（Browne 等人，2012）或进化算法（Bäck 等人，1997年）实现。

5. 实验评估

我们评估语义数据算子作为专家编写和智能体生成的机器学习流水线中一等的、可复用的基本构件。按照设计，SEMPIPES 既不是自动化机器学习系统，也不是端到端的智能体流水线生成器。相反，SEMPIPES 引入了一种编程抽象，用于合成和优化机器学习流水线中的复杂数据操作。用于评估预测任务的全自动流水线生成基准测试（例如 MLEBench（Chan 等人，2025年））与我们的设置存在根本性的不匹配。此外，表格机器学习流水线涵盖的任务范围远不止预测，还包括数据集成、清洗、特征提取、调试和增强，而现有基准对这些任务的覆盖并不充分。我们设计了专门的实验，以捕捉这些以数据为中心的挑战，并在现实且复杂的流水线中隔离语义算子的影响。所有实验均使用谷歌的`gemini-2.5-flash`型号，非优化执行的温度设为 0.0，优化过程的温度设为 2.0。

5.1. 语义数据算子改进并简化表格机器学习流水线

我们展示了，经过优化的语义数据算子能够改进专家编写和智能体生成的机器学习流水线。

实验设置。我们在一组具有挑战性的以数据为中心的任务上评估 SEMPIPES，这些任务涵盖了多种机器学习应用领域。任务涉及复杂的数据集成、清洗、特征提取与生成、调试和数据增强，选自近期发表于 NeurIPS（Chen 等，2023年；Qian 等人，2023年）、EMNLP（Lee 等人，2021）、SIGMOD 编程竞赛（Ives 等人，2022年）以及各类 Kaggle 竞赛中的工作。我们选择的任务具有高度异质性，例如，输入数据的规模从数千条记录到数百万条记录不等，其中若干任务涉及多个输入表，或将表格数据与文本或图像结合起来。我们从相关研究论文附带的代码仓库、SIGMOD 编程竞赛的顶尖参赛作品以及 Kaggle 获奖者的笔记本中收集了专家手工编写的流水线，这些成果体现了丰富的领域知识和大量的工程投入。此外，我们使用 AIDE（*Jiang* 等，2025a）和 mini-SWE-agent（Yang 等，2024b）合成了智能体生成的流水线（Yang 等，尽管经过大量提示和重试，这些智能体中仍至少有一个无法为七项任务中的六项生成可运行或正确的流水线，这凸显了我们所选任务的复杂性。性。我们的实验协议如下：（1）首先，我们在每项任务上评估原始的专家流水线和智能体流水线；（2）然后，将它们改写为 skrub DataOps 流水线，并引入 SEMPES语义算子，同时为这些算子提供一条自然语言指令，用以概述原始流水线中的专家知识。请注意，我们固定模型选择和超参数，以隔离实际数据操作的影响。我们评估改写后的流水线，然后（3）对其语义算子进行 24 轮优化。每条流水线均重复运行五次；有关各项任务的起源、评估指标和设置的详细信息，请参阅A.6 节。

结果与讨论。表 2 总结了对 19 条评估流水线的结果，包括特定任务的指标以及每条流水线的来源（专家编写或智能体生成）。结果证实，语义数据算子能够提升专家编写和智能体生成的流水线的预测性能。在 19 个案例中的 17 个案例里，即使不进行优化，用语义算子和概述专家知识的指令（“使用 SemOps 和指令”）扩展原始流水线也能提升预测性能。在一个案例中，性能保持不变（kaggle-movie-a）；另一个案例（sustech）中，召回率略微下降了 0.001，我们认为这是因为原始代码使用了微调模型，而我们没有重新调优该模型。优化语义数据算子（“使用优化后的 SemOps”）能带来进一步提升。在 19 个案例中，有 18 个案例的优化后流水线优于原始代码（“原始流水线”）；在 19 个案例中的 17 个案例里，优化后流水线的表现也优于使用未优化语义算子的流水线。我们讨论三个典型案例（更多详情见第 A.6.3 节）。首先，在通过特征工程预测拼字游戏玩家评分的任务中，我们观察到专家编写和智能体生成的流水线均取得了显著提升：使用优化后语义算子的流水线使均方根误差降低了 18% 至 26%。我们认为，这是因为复杂的输入数据（包含 3 张表，涉及约 7 万名玩家；这些玩家共参与了 8 万场游戏，累计 200 万个回合）为构建有用特征提供了丰富的机会。我们通过指令让语义算子生成与玩家统计数据相关、有助于预测玩家评分的特征。基于此，SEMPIPES 改进了 kaggle-scrabble 流水线（该流水线使用带强正则化的 XGBoost 模型，并包含大量浅层树），例如计算了 100 多个额外特征，包括玩家先手时的胜率、游戏节奏或得分回合效率。其次，我们将 SEMPIPES 应用于 SIGMOD 2022 编程竞赛任务。该任务要求在包含数百万条记录的数据集上大规模执行特征提取和实体分块。实体分块用于识别并关联指向同一现实世界实体的不同记录。我们重写了竞赛中排名第五的解决方案（rutgers），该方案针对更具挑战性的多

表 2：**在专家编写和智能体生成的流水线中，使用带有自然语言指令的语义数据算子处理广泛的以数据为中心的任务所得到的结果**。在 19 个案例中的 18 个里，语义算子相较原始代码提升了性能。所有进步均仅来自数据操作，因为我们的实验中原始代码的模型选择和超参数保持不变。

以数据为中心的任务	指标	流水线	起源 专家	起源 智能体	原始流水线	使用 SemOps 和指令	使用优化后的 SemOps
临床提取信息增强特征	F1分数 ↑	micromodels	✓		0.665 ± 0.01	0.692 ± 0.02	**0.729** ± 0.00
		软件工程微模型		✓	0.690 ± 0.00	0.692 ± 0.02	**0.729** ± 0.00
阻塞适用于实体解析	召回率 ↑	rutgers	✓		0.238 ± 0.00	0.256 ± 0.01	**0.264** ± 0.01
		sustech	✓		**0.322** ± 0.00	0.320 ± 0.00	0.321 ± 0.00
		AIDE-SIGMOD		✓	0.043 ± 0.00	0.163 ± 0.03	**0.165** ± 0.03
		SIGMOD 基线	✓		0.099 ± 0.00	0.141 ± 0.02	**0.173** ± 0.05
特征工程(电影收入)	均方根对数误差 ↓	kaggle-movie-a	✓		2.222 ± 0.14	2.222 ± 0.14	**2.203** ± 0.12
		kaggle-movie-b	✓		2.170 ± 0.07	2.129 ± 0.15	**2.122** ± 0.14
		AIDE 电影		✓	2.355 ± 0.14	2.319 ± 0.15	**2.269** ± 0.20
特征工程(房价)	均方根对数误差 ↓	kaggle-house-a	✓		0.158 ± 0.03	**0.149** ± 0.00	0.157 ± 0.03
		kaggle-house-b	✓		0.159 ± 0.03	0.153 ± 0.03	**0.150** ± 0.03
		aide-house		✓	0.165 ± 0.04	0.163 ± 0.05	**0.163** ± 0.04
		swe-house		✓	0.177 ± 0.04	0.168 ± 0.04	**0.164** ± 0.03
特征工程(Scrabble评分)	均方根误差 ↓	kaggle-scrabble	✓		176.1 ± 24.7	149.2 ± 24.8	**144.4** ± 18.3
		AIDE拼字游戏		✓	255.0 ± 24.9	195.2 ± 21.9	**186.7** ± 21.1
		SWE拼字游戏		✓	228.5 ± 12.9	210.1 ± 43.0	**187.8** ± 21.4
数据标注	准确率 ↑	hibug	✓		0.598 ± 0.05	0.758 ± 0.15	**0.766** ± 0.16
数据增强为了公平性	曲线下面积 ↑	sivep	✓		0.653 ± 0.03	0.682 ± 0.00	**0.710** ± 0.04
		swe-fair		✓	0.650 ± 0.04	0.682 ± 0.02	**0.710** ± 0.04

语数据集，而原始方法的召回率较低。使用 SEMPIPES，我们用对 `sem_extract_特征` 和 `sem_gen_特征` 的调用，替代了数百行手写的特征提取与阻塞代码，从而丰富了特征集。我们的优化可使 `rutgers` 在公开排行榜上的排名从第五升至第三。第三，在模型调试任务的数据标注中，使用 `Shibug` 流水线，并通过 SEMPIPES 优化专家编写的流水线后，标注准确率提升了 16% 以上。我们认为，这一进步归功于优化器选用了效果良好的预训练模型。此外，我们还发现语义数据算子能够简化专家编写的流水线，详见第 A.6.4 节。

总之，SEMPIPES 能够优化机器学习流水线中的数据操作；在以往仅关注模型和超参数的工作中，这些数据操作通常是固定的。我们在具有强大专家基线方法的挑战性任务上的结果表明，优化数据操作为提升表格机器学习性能提供了一条额外途径。所有增益都仅来自数据操作，因为在我们的实验中，模型选择和超参数保持不变。基于大语言模型的实验可能会受到训练时污染的影响，例如在预训练期间接触到任务的解决方案。我们认为这不太可能影响我们的结果：智能体生成的流水线在我们的任务上表现不佳或失败；若干任务并非源自现有竞赛；而且语义算子已通过其自然语言指令显式编码了领域知识。

5.2. 语义数据算子的表现与专用方法持平或更优

我们验证，SEMPIPES 算子通过与强基线方法的比较，其性能达到或超过专用方法。参见第 A.7 节 了解更多细节。

实验设置。对于使用 sem_gen_features 进行特征生成，我们将其与自动化特征工程方法 CAAFE（Hollmann 等，2023b）进行比较。我们使用 CAAFE 作者提供的实验设置和预先计算的特征工程代码，在三个数据集上进行评估。我们采用随机森林和未经调优的 TabPFN 2.0（Hollmann 等，2023a）作为下游模型，并报告测试划分上的平均曲线下面积。对于 SEMPIPES，我们在留出数据上优化语义流水线。对于使用 sem_fillna 进行缺失值填补，我们在论文中的两个基准任务上与 Narayan 等人（2022）的提示方案进行比较，并测量填补值的准确率。对于使用 sem_extract_features 进行零样本特征提取，我们将 SEMPIPES 合成的代码与 SEMPIPES 中直接逐项进行大语言模型处理的方式，以及语义查询处理系统 Palimpzest（Liu 等，2025）和 LOTUS（Patel 等，2025）进行比较。我们在文本（金融条款分类（Alrashidan，2022））、图像（X 射线肺炎检测（Kermany 等，2018））和音频（环境声音提取（Piczak，2015））上进行评估，每种模态处理 1,000 和 10,000 个样本。

表格 3：在三个数据集上，不进行特征工程（No FE）以及使用随机森林（RF）和 TabPFN 进行自动化特征工程时的 AUROC 分数。

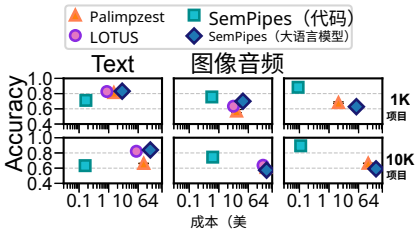
	数据集	无特征工程	CAAFE	SemPipes
RF	平衡-刻度	0.685 ± 0.13	0.737 ± 0.19	**0.884** ± 0.23
	井字棋圈	0.807 ± 0.07	0.878 ± 0.11	**0.929** ± 0.08
	航空公司	0.629 ± 0.03	**0.631** ± 0.03	0.625 ± 0.03
TabPFN	平衡-刻度	0.848 ± 0.25	0.863 ± 0.24	**0.938** ± 0.11
	滴答声 toe	0.959 ± 0.02	0.945 ± 0.07	**0.997** ± 0.00
	航空公司	0.647 ± 0.03	0.647 ± 0.03	**0.650** ± 0.03

表4：使用 SemPipes 和 Narayan 等人，两种方法均采用直接大语言模型处理，并使用十个少样本示例。

缺失值填补

数据集	Narayan 等人	SemPipes
Restaurant	0.761 ± 0.00	**0.827** ± 0.01
Buy	0.923 ± 0.00	**0.988** ± 0.01

图3：零样本特征提取。



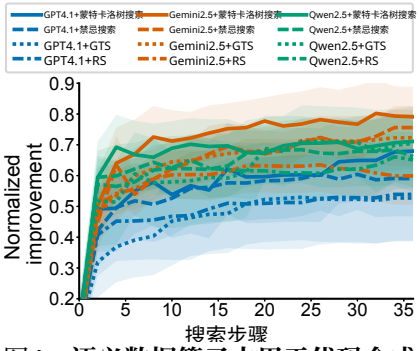


图4：**语义数据算子中用于代码合成的大语言模型优化效果**。搜索策略包括蒙特卡洛树搜索（MCTS）、截断选择（TS）、贪心树搜索（GTS）或随机搜索（RS）。

结果与讨论。表3展示了特征工程的结果：在三个数据集中的两个上，CAAFE 和 SEMPIPES 均显著提升了两个下游模型的性能。CAAFE 使随机森林（RF）的性能略有提升（4.2%），对 TabPFN 的提升则微乎其微（0.03%）。SEMPIPES 带来的提升幅度大得多：与不进行特征工程相比，随机森林的性能平均提高了 10.6%，TabPFN 提高了 4.4%。我们认为，这一优势源于引导式树结构优化，而非 CAAFE 的贪心线性搜索，也源于 SEMPIPES 能够针对下游流水线定制提示词，例如确定所用模型的类型。对于缺失值插补（表4），SEMPIPES 在两个数据集上均优于人工设计的提示基线，在 Restaurant 上提升了 6.3%，在 Buy 上提升了 1.6%。这一进步得益于语义算子的设计：它们将自动计算的任务特定统计量（如唯一值和列类型）与定制的算子指令相结合，其中包含更多有效的少样本示例，并以人类可读的格式呈现。对于零样本特征提取（图3），SEMPIPES 的代码生成在图像和音频模态上均取得了最佳性能。SEMPIPES 始终优于 Palimpzest 和 LOTUS，音频准确率提高了 ~20%，图像准确率提高了~10%，同时成本效益显著更高。对于文本，SEMPIPES 实现了清晰的精度–成本权衡：直接使用大语言模型处理可获得最高准确率（略高于 Palimpzest 和 LOTUS，这得益于更精细的提示词），而代码生成的准确率低 10%，但成本仅为其十分之一，从而能够以固定成本实现可扩展推理。SEMPIPES 不依赖逐样本的大语言模型推理，而是合成可执行代码，调用预训练的领域专用模型，例如用于文本的 DeBERTa（He 等，2021）、用于图像的 BiomedCLIP（Zhang 等，2024）和用于音频的 CLAP（Wu 等，2023），从而实现可扩展且高成本效益的推理。当从 1,000 个对于 10,000 个数据项目，SEMPIPES 中的代码合成只需恒定的代码生成开销，而大语言模型的处理成本会线性增长，因此高出十倍。

5.3. 不同大语言模型在语义数据算子代码合成中的优化效果

我们使用基于 skrub 项目提供的数据集构建的分类流水线基准测试，评估大语言模型和搜索策略对语义算子的优化效果（第 A.8 节）。我们测试了 GPT-4.1、Gemini-2.5-flash 和 Qwen-Coder-2.5 (32B) 作为代码合成器，并分别搭配蒙特卡洛树搜索（MCTS）（Browne 等人，2012）、截断选择（Bäck 等人，1991年）、贪心树搜索（Jiang 等人，2025a）或随机搜索。如图 4 所示，所有模型都提升了预测性能，大部分提升在五到八个搜索步骤内实现，这表明只需适度的预算便可实现显著优化。在各种搜索策略中，MCTS 的表现优于其他策略，我们认为这归功于它能够像相关研究中那样调节探索与利用（Toledo 等人，2025年）。

6. 结论

我们提出了 SemPipes，这是一种编程模型，可将表格机器学习流水线中的数据操作指定为语义算子，并通过进化搜索引导的大语言模型代码合成来优化这些操作。我们当前方法的一个局限是尚不支持时间序列或视频。此外，执行由大语言模型合成的代码存在潜在安全风险，在实际应用中需要通过在强化安全的沙箱中运行来缓解（netblue30，2022年）。未来，我们计划添加更多语义数据算子，例如语义连接，并探索它们作为智能体原语在 AIDE（Jiang 等人，2025a）等系统中的价值。此外，我们将扩展优化器，以联合优化预测性能与执行时间或成本。

参考文献

Agrawal、L. A.、Tan、S.、Soylu、D.、Ziems、N.、Khare、R.、Opsahl-Ong、K.、Singhvi、A.、Shandilya、H.、Ryan、M. J.、Jiang、M.、Potts、C.、Sen、K.、Dimakis、A. G.、Stoica、I.、Klein、D.、Zaharia、M. 和 Khattab、O. G. Gepa：反思式提示词演化可以胜过强化学习，2025年。网址：https://arxiv.org/abs/2507.19457。

Alrashidan, M. 法律条款。https://www.kaggle.com/datasets/mohammedalrashidan/contracts-clauses-datasets, 2022年。Kaggle。

Anaconda. 2022年数据科学现状报告：为创新铺平道路。技术报告，Anaconda, Inc.，2022年。网址 https://know.anaconda.com/rs/387-XNW-688/images/ANA_2022SODSReport.pdf。访问日期：2025-11-16。

Anthropic。Claude Code：人工智能代码助手。https://www.claude.com/product/claude-code，2025年。访问日期：2026-01-27。Anysphere。Cursor：人工智能代码编辑器。https://www.cursor.com/，2025年。访问日期：2026-01-27。Aygün、E.、Belyaeva、A.、Comanici、G.、Coram、M.、Cui、H.、Garrison、J.、Kast、R. J. A.、McLean、C. Y.、Norgaard、P.、Shamsi、Z. 等。帮助科学家编写专家级实证软件的人工智能系统，2025年。网址 https://arxiv.org/abs/2509.06503。Bäck、T.、Hoffmeister、F. 和 Schwefel、H. 进化策略综述。载于 Belew、R. K.和Booker、L. B.（编），第四届国际遗传算法会议论文集，美国加利福尼亚州圣迭戈，*1991*年7月，第2–9页。Morgan Kaufmann，1991年。网址 https://dblp.org/rec/conf/icga/BackHS91.bib。Bäck、T.、Fogel、D. B. 和 Michalewicz、Z.进化计算手册。IOP Publishing Ltd.，英国，第一版，1997年。ISBN 0750303921。网址 https://dl.acm.org/doi/book/10.5555/548530。Baqui、P.、Marra、V.、Alaa、A. M.、Bica、I.、Ercole、A. 和 van der Schaar、M. 使用机器学习比较巴西的COVID-19风险因素：社会经济、人口统计和结构性因素的重要性。《科学报告》，11(1):15591，2021年8月。ISSN 2045-2322。doi：10.1038/s41598-021-95004-8。网址 https://doi.org/10.1038/s41598-021-95004-8。Baylor、D.、Breck、E.、Cheng、H.-T.、Fiedel、N.、Foo、C. Y.、Haque、Z.、Haykal、S.、Ispir、M.、Jain、V.、Koc、L.、Koo、C. Y.、Lew、L.、Mewald、C.、Modi、A. N.、Polyzotis、N.、Ramesh、S.、Roy、S.、Whang、S. E.、Wicke、M.、Wilkiewicz、J.、Zhang、X. 和 Zinkevich、M. TFX：基于TensorFlow的生产规模机器学习platform。In *SIGKDD International Conf*知识发现与数据挖掘会议（SIGKDD），第1387–1395页。计算机协会，2017年。ISBN 9781450348874。doi：10.1145/3097983.3098021。URL https://doi.org/10.1145/3097983.3098021。Becker, J.、Rush, N.、Barnes, E. 和 Rein, D. 衡量2025年初的人工智能对资深开源开发者生产力的影响，2025年。URL https://arxiv.org/abs/2507.09089。Belkhiter, Y.、Tirupathi, S.、Zizzo, G.、Sharma, S. 和 Kelleher, J. D. 自动化机器学习中的事前预测：利用大语言模型改进表格数据集的模型选择与基准测试，2025年。URL https://arxiv.org/abs/2510.01842。Browne, C. B.、Powley, E.、Whitehouse, D.、Lucas, S. M.、Cowling, P. I.、Rohlfshagen, P.、Tavener, S.、Perez, D.、Samothrakis, S. 和 Colton, S. 蒙特卡洛树搜索方法综述。*IEEE* 计算智能与游戏人工智能汇刊，4(1):1–43，2012年。doi：10.1109/TCIAIG.2012.2186810。URLhttps://doi.org/10.1109/TCIAIG.2012.2186810。Cappuzzo, R.、Coelho, A.、Lefebvre, F.、Papotti, P. 和 Varoquaux, G. 检索、合并、预测：利用数据湖扩充表格。机器学习研究汇刊，2025年。ISSN 2835-8856。URLhttps://openreview.net/forum?id=4uPJN6yfY1。Chan, J. S.、Chowdhury, N.、Jaffe, O.、Aung, J.、Sherburn, D.、Mays, E.、Starace, G.、Liu, K.、Maksin, L.、Patwardhan, T.、Madry, A. 和 Weng, L. MLE-bench：评估机器学习工程领域的机器学习智能体。载于第十三届国际学习表征会议，2025年。URL https://openreview.net/forum?id=6s5uXNWGIh。Chen, M.、LI, Y. 和 Xu, Q. HiBug：面向人类可解释模型调试。载于神经信息处理系统进展，第 36 卷，第 4753–4766 页。Curran Associates, Inc.，2023年。URL https://proceedings.neurips.cc/paper_file/p a p e r/2023/file/0f53ecc0d36a5d5d3d3e94d42c4b23ca-Paper-Conference.pdf。Databricks。使用 Databricks AI函数对数据应用人工智能，2025年。URL https://docs.databricks.com/aws/en/large-language-models/ai-functions。最后更新于2025年12月19日。访问日期：2025年1月27日。Dorbani, A.、Yasser, S.、Lin, J. 和 Mhedhbi, A. 超越鸭叫：语言模型与检索增强生成在 DuckDB 中的深度集成。*VLDB*基金会会刊，18(12):5415–5418，2025年8月。ISSN 2150-8097。doi：10.14778/3750601.3750685。URLhttps://doi.org/10.14778/3750601.3750685。

Fang, H.、Han, B.、Erickson, N.、Zhang, X.、Zhou, S.、Dagar, A.、Zhang, J.、Turkmen, A. C.、Hu, C.、Rangwala, H.、Wu, Y. N.、Wang, B. 和 Karypis, G. MLZero：一种端到端机器学习自动化多智能体系统。载于第三十九届神经信息处理系统年会，2025年。URLhttps://openreview.net/forum?id=9t7gfYUp0U。

Fathollahzadeh, S.、Mansour, E.与Boehm, M. Catdb：数据目录引导的、基于大语言模型的以数据为中心的机器学习流水线生成。*VLDB基金会会刊*，18(8):2639–2652，2025年。ISSN 2150-8097。doi: 10.14778/3742728.3742754。网址 https://doi.org/10.14778/3742728.3742754。

Feurer, M.与Hutter, F. 超参数优化，第3–33页。施普林格国际出版社，Cham，2019年。ISBN 978-3-030-05318-5。doi: 10.1007/978-3-030- 05318-5_1。网址 https://doi.org/10.1007/978-3-030-05318-5_1。

Fey, M., Kocijan, V., Lopez, F., Lenssen, J. E., 和 Leskovec, J. Kumorfm：一种用于关系数据上下文学习的基础模型，2025年。网址 https://kumo.ai/company/news/kumo-relational-foundation-model/。访问日期：2026-01- 27。

GitHub。GitHub Copilot：你的 AI结对程序员，2025年。网址https://github.com/features/copilot。访问日期：2025-11-06。

谷歌。BigQuery 中的生成式人工智能简介，2025年。网址 https://docs.cloud.google.com/bigquery/docs/generative-ai-overview。最后更新于 2025-01-23 UTC。访问日期：2026-01-27。

Grafberger, S., Groth, P., 和 Schelter, S. 在原生机器学习流水线上自动化并优化以数据为中心的假设分析。*ACM 数据管理会议论文集*，1(2)，2023年6月。doi：10.1145/3589273。网址 https://doi.org/10.1145/3589273。

Grinsztajn, L., Oyallon, E. 和 Varoquaux, G. 为什么基于树的模型在典型表格数据上仍优于深度学习？载于 第三十六届神经信息处理系统会议数据集与基准赛道，2022年。网址：https://openreview.net/forum?id=Fp7__phQszn。

Grinsztajn, L., Flöge, K., Key, O., Birkel, F., Jund, P., Roof, B., Jäger, B., Safaric, D., Alessi, S., Hayler, A., Manium, M., Yu, R., Jablonski, F., Hoo, S. B., Garg, A., Robertson, J., Bühler, M., Moroshan, V., Purucker, L., Cornu, C., Wehrhahn, L. C., Bonetto, A., Schölkopf, B., Gambhir, S., Hollmann, N., 以及 Hutter, F. Tabpfn-2.5：推动表格基础模型领域的最先进水平，2025年。链接 https://arxiv.org/abs/2511.08667。

He, P., Liu, X., Gao, J., 以及 Chen, W. Deberta：采用解耦注意力的增强解码 BERT。在 国际学习表征会议，2021年。链接 https://openreview.net/forum?id=XPZIaotutsD。

Hollmann, N.、Müller, S.、Eggensperger, K. 和 Hutter, F. TabPFN：一种能在一秒内解决小型表格分类问题的 Transformer。载于 第十一届国际学习表征会议，2023a年。网址：https://openreview.net/forum?id=cp5PvcI6w8_。

Hollmann, N.、Müller, S. 和 Hutter, F. 用于自动化数据科学的大语言模型：介绍面向上下文的自动化特征工程工具 CAAFE。 神经信息处理系统进展，36：44753–44775，2023b年。网址：https://openreview.net/forum?id=9WSxQZ9mG7。

Hollmann, N., Müller, S., Purucker, L., Krishnakumar, A., Körfer, M., Hoo, S. B., Schirrmeister, R.T., 和 Hutter, F. 使用表格基础模型在小规模数据上实现准确预测。《自然》，637(8045):319–326，2025年。doi: 10.1038/s41586-024-08328-6。网址 https://doi.org/10.1038/s41586-024-08328-6。

Ives, Z.G., Bonifati, A., 和 Abbadi, A.E.（编）。*SIGMOD '22*：国际数据管理会议，费城，美国，*2022*年*6*月*12*日至*17*日，2022年。ACM。ISBN 978-1-4503-9249-5。doi: 10.1145/3514221。网址https://doi.org/10.1145/3514221。

Jiang, Z., Schmidt, D., Srikanth, D., Xu, D., Kaplan, I., Jacenko, D., 和 Wu, Y. AIDE：代码空间中的人工智能驱动探索，2025a。网址 https://arxiv.org/abs/2502.13138。

Jiang, Z., Schmidt, D., Srikanth, D., Xu, D., Kaplan, I., Jacenko, D., 和 Wu, Y. Aide：示例。https://github.com/WecoAI/aideml/tree/main/sample_结果，2025b。

Jo, S. 和 Trummer, I. ThalamusDB：多模态数据的近似查询处理。*ACM Manag. Data* 论文集，2(3)，2024。doi: 10.1145/3654989。网址https://doi.org/10.1145/3654989。

Karlaš, B., Dao, D., Interlandi, M., Schelter, S., Wu, W. 和 Zhang, C. 基于端到端机器学习流水线上的 Shapley 重要性进行数据调试。在第十二届机器学习表示国际会议，2024 年。网址：https://openreview.net/forum?id=qxGXjWxabq。

Karpathy, A. 软件正在改变（再次）。https://www.youtube.com/watch?v=LCEmiRjPEtQ，2025年。

Kermany, D. S., Goldbaum, M., Cai, W., Valentim, C. C., Liang, H., Baxter, S.L., McKeown, A., Yang, G., Wu, X., Yan, F., Dong, J., Prasadha, M. K., Pei, J., Ting, M. Y., Zhu, J., Li, C., Hewett, S., Dong, J., Ziyar, I., Shi, A., Zhang, R., Zheng, L., Hou, R., Shi, W., Fu, X., Duan, Y., Huu, V. A., Wen, C., Zhang, E. D., Zhang, C. L., Li, O., Wang, X., Singer, M. A., Sun, X., Xu, J., Tafreshi, A., Lewis, M. A., Xia, H., and Zhang, K. 通过基于图像的深度学习识别医学诊断和可治疗疾病。《细胞》，172(5):1122–1131.e9，2018年2月。ISSN 0092-8674。doi: 10.1016/j.cell.2018.02.010。URLhttps://doi.org/10.1016/j.cell.2018.02.010。

Khattab, O.、Singhvi, A.、Maheshwari, P.、Zhang, Z.、San- thanam, K.、A, S. V.、Haq, S.、Sharma, A.、Joshi, T. T.、Moazam, H.、Miller, H.、Zaharia, M. 和 Potts, C. DSPy：将声明式语言模型调用编译为先进的管道。载于第十二届国际学习表征会议，2024年。网址：https://openreview.net/forum?id=sY5N0zY5Od。

Küken, J.、Purucker, L. 和 Hutter, F. 大语言模型为表格数据生成过多简单特征，2025年。网址：https://arxiv.org/abs/2410.17787。

Lao, J.、Zimmerer, A.、Ovcharenko, O.、Cong, T.、Russo, M.、Vitagliano, G.、Cochez, M.、Özcan, F.、Gupta, G.、Hottelier, T.、Jagadish, H. V.、Kissel, K.、Schelter, S.、Kipf, A. 和 Trummer, I. SemBench：语义查询处理引擎的基准测试，2025年。网址：https://arxiv.org/abs/2511.01716。

Lee, A., Kummerfeld, J. K., An, L., 和 Mihalcea, R. 面向高效、可解释且可复用系统的迷你模型：心理健康案例研究。载于 计算语言学协会发现论文集：*EMNLP 2021*，第4257–4272 页，蓬塔卡纳，多米尼加共和国，2021 年。计算语言学协会。doi: 10.18653/v1/2021.findings-emnlp.360。网址：https://aclanthology.org/2021.findings-emnlp.360/。

Lee, Y.-A., Yi, G.-T., Liu, M.-Y., Lu, J.-C., Yang, G.-B., 和 Chen, Y.-N. 复合人工智能系统优化：方法、挑战与未来方向综述。载于 Christodoulopoulos, C., Chakraborty, T., Rose, C., 和 Peng, V.（编），*2025*年自然语言处理经验方法会议论文集，第 28760–28775 页。计算语言学协会，2025年。ISBN 979-8-89176-332-6。doi：10.18653/v1/2025. emnlp-main.1463。网址：https://aclanthology.org/2025.emnlp-main.1463/。

Liskowski, P., Han, B., Aggarwal, P., Chen, B., Jiang, B., Jindal, N., Li, Z., Lin, A., Schmaus, K., Tayade, J., Zhao, W., Datta, A., Wiegand, N., 和 Tsirogiannis, D. Cortex AISQL：面向非结构化数据的生产级 SQL引擎，2025年。网址 https://arxiv.org/abs/2511.07663。

Liu, C., Russo, M., Cafarella, M., Cao, L., Chen, P. B., Chen, Z., Franklin, M., Kraska, T., Madden, S., Shahout, R., 和 Vitagliano, G. Palimpzest：通过声明式查询处理优化 AI 驱动的分析。在创新数据库研究会议（*CIDR*）论文集，2025年。网址 https://vldb.org/cidrdb/papers/2025/p12-liu.pdf。

Meng, X.、Bradley, J.、Yavuz, B.、Sparks, E.、Venkatraman, S.、Liu, D.、Freeman, J.、Tsai, D.、Amde, M.、Owen, S.、Xin, D.、Xin, R.、Franklin, M. J.、Zadeh, R.、Zaharia, M. 与 Talwalkar, A. MLlib：Apache Spark 中的机器学习。《机器学习研究杂志》，17 (1):1235–1241，2016年。ISSN 1532-4435。网址https://jmlr.org/papers/v17/15-237.html。

Nam, J.、Yoon, J.、Chen, J. 与 Pfister, T. DS-STAR：通过迭代规划与验证实现的数据科学智能体，2025a。网址 https://arxiv.org/abs/2509.21825。

Nam, J.、Yoon, J.、Chen, J.、Shin, J.、Arik, S. O. 与 Pfister, T. MLE-STAR：通过搜索与定向优化实现的机器学习工程智能体。载于第三十九届神经信息处理系统年会，2025b。网址 https://openreview.net/forum?id=vS1M06Px6u。

Narayan, A.、Chami, I.、Orr, L. 和 Ré, C. 基础模型能整理你的数据吗？*VLDB*基金会会刊，16（4）：738–746，2022年12月。ISSN 2150-8097。doi：10.14778/3574245.3574258。网址：https://doi.org/10.14778/3574245.3574258。

netblue30。Firejail：Linux 命名空间与 seccomp-bpf 沙箱。https://github.com/netblue30/firejail，2022年。GitHub 仓库。

Nguyen, T.、Yates, A.、Zirikly, A.、Desmet, B. 和 Co- han, A. 通过利用临床问卷提升抑郁症检测的泛化能力。载于计算语言学协会第*60*届年会论文集（第*1*卷：长篇论文），第8446–8459页，爱尔兰都柏林，2022年5月。计算语言学协会。doi: 10.18653/v1/2022. acl-long.578。网址：https://aclanthology.org/2022.acl-long.578/。

Nolan, B. 人工智能驱动的编程工具抹掉了一家软件公司的数据库，造成“灾难性故障”。https://fortune.com/2025/07/23/ai-coding-tool-replit-wiped-database-called-it-a-catastrophic-failure/，2025年7月23日。

Opsahl-Ong, K., Ryan, M. J., Purtell, J., Broman, D., Potts, C., Zaharia, M., and Khattab, O. 优化多阶段语言模型程序的指令和示例。载于 Al-Onaizan, Y., Bansal, M., and Chen, Y.-N.

（编者），*2024*年自然语言处理实证方法会议论文集，第 9340–9366 页。计算语言学协会，2024 年 11 月。doi: 10.18653/v1/2024.emnlp-main.525。网址：https://aclanthology.org/2024.emnlp-main.525/。

Patel, L.、Jha, S.、Pan, M.、Gupta, H.、Asawa, P.、Guestrin, C. 和 Zaharia, M. 语义算子及其优化：借助准确性保证，在 LOTUS 中实现基于大语言模型的数据处理。*VLDB*基金会会刊，2025年。网址：https://doi.org/10.14778/3749646.3749685。

Pedregosa, F.、Varoquaux, G.、Gramfort, A.、Michel, V.、Thirion, B.、Grisel, O.、Blondel, M.、Prettenhofer, P.、Weiss, R.、Dubourg, V.、Vanderplas, J.、Passos, A.、Cournapeau, D.、Brucher, M.、Perrot, M. 和 Édouard Duchesnay。scikit-learn：Python 中的机器学习。《机器学习研究杂志》，12(85)：2825–2830，2011年。网址：http://jmlr.org/papers/v12/pedregosa11a.html。

Petersohn, D., Macke, S., Xin, D., Ma, W., Lee, D., Mo, X., Gonzalez, J. E., Hellerstein, J. M., Joseph, A. D., and Parameswaran, A. 面向可扩展的数据框系统。*VLDB*基金会会刊，13(12)，2020年。ISSN 2150-8097。doi: 10.14778/3407790.3407807。URLhttps://doi.org/10.14778/3407790.3407807。

Piczak, K. J. ESC：用于环境声音分类的数据集。载于第*23*届*ACM*多媒体年度会议论文集，第1015–1018页。ACM Press，2015年。ISBN 978-1-4503-3459-4。doi: 10.1145/2733373.2806390。URLhttp://dl.acm.org/citation.cfm?doid=2733373.2806390。

Polyzotis, N.、Roy, S.、Whang, S. E. 和 Zinkevich, M. 生产环境中的机器学习中的数据管理挑战。在*2017*年*ACM* 数据管理国际会议论文集，SIGMOD '17，第 1723–1726 页，2017年。ISBN 9781450341974。doi: 10.1145/3035918.3054782。URLhttps://doi.org/10.1145/3035918.3054782。

Polyzotis, N., Zinkevich, M., Roy, S., Breck, E., and Whang, S. 机器学习的数据验证。*MLSYS*，1:334–347，2019年。网址https://proceedings.mlsys.org/paper_files/paper/2019/hash/928f1160e52192e3e0017fb63ab65391-Abstract.html。

Psallidas, F., Zhu, Y., Karlas, B., Henkel, J., Interlandi, M., Krishnan, S., Kroth, B., Emani, V., Wu, W., Zhang, C., 等。透过镜头看数据科学：对数百万个 GitHub 笔记本和 ML.NET 流水线的分析。*ACMSIGMOD Record*，51(2):30–37，2022年。ISSN 0163-5808。doi: 10.1145/3552490.3552496。网址https://doi.org/10.1145/3552490.3552496。

Qian, Z., Davis, R., and van der Schaar, M. Synthcity：适用于表格合成数据多种用例的基准测试框架。*NeurIPS*，36:3173–3188，2023年。网址https://proceedings.neurips.cc/paper_files/paper/2023/hash/09723c9f291f6056fd1885081859c186-Abstract-Datasets_和_Benchmarks.html。

Qu, J.、Holzmüller, D.、Varoquaux, G. 和 Morvan, M.L. Tabicl：一种用于在大型数据上进行上下文学习的表格基础模型。在第四十二届国际机器学习会议，2025年。URL https://openreview.net/forum?id=0VvD1PmNzM。

Reddit。[D] 如今，高级机器学习工程是否只需要调用 API？reddit，2023年。URLhttps://www.reddit.com/r/MachineLearning/comments/1npdfh1/d_is_senior_ml_engineering_just_api_calls_now/。r/MachineLearning，访问日期：2025年-10-05。

Russo, M. 和 Kraska, T. Deep Research 是新一代分析系统：迈向构建人工智能驱动分析的运行时环境。*CIDR*，2026年。网址https://www.cidrdb.org/papers/2026/p30-russo.pdf。

Santos, I. D.A. B., DaSilva, M.H. L. F., Rodrigues, M. G. D. A., Simão, E. P., Lynn, T., Sampaio, V. D. S., Endo, P. T. 等。用于支持通过 COVID-19 和流感鉴别诊断严重急性呼吸综合征（SARS）的巴西分类数据集。载于*2023*年伊比利亚第*18*届信息系统与技术会议（*CISTI*），第 1–4 页。IEEE，2023年。doi: 10.23919/CISTI58278.2023.10211694。网址https://doi.org/10.23919/CISTI58278.2023.10211694。

Schäfer, B., Purucker, L., Janowski, M.和Hutter, F. 自动化特征工程对表格数据的可用性如何？*arXiv*预印本*arXiv:2508.13932*，2025年。网址https://doi.org/10.48550/arXiv.2508.13932。

scikit-learn。管道与复合估计器。https://scikit-learn.org/stable/api/sklearn.compose.html，2025年。访问日期：2025-11-16。Shi, C.、Yang, K.、Chen, Z.、Li, J.、Yang, J.和Shen, C. 从最佳臂识别的视角进行高效提示优化。载于第三十八届神经信息处理系统年会，2024。网址https://openreview.net/forum?id=FLNnlfBGMo。Sinha, A.、Malo, P.和Deb, K. 双层优化综述：从经典方法到进化方法及其应用。*IEEE*进化计算汇刊，22(2):276–295，2017年。doi: 10.1109/TEVC. 2017.2712906。网址https://doi.org/10.1109/TEVC.2017.2712906。

skrub。使用数据操作的复杂多表管道，2025a。网址 https://skrub-data.org/stable/data_ops.html。访问日期：2025-11-09。

skrub。skrub 数据操作与其他方案有何不同? https://skrub-data.org/stable/modules/data_ops/basics/data_ops_vs_alternatives.html，2025b。访问日期：2025-12-09。

pandas 开发团队。pandas-dev/pandas：Pan- das，2020年2月。网址https://doi.org/10.5281/zenodo.3509134。

Toledo, E., Hambardzumyan, K., Josifoski, M., Hazra, R., Baldwin, N., Audran-Reiss, A., Kuchnik, M., Magka, D., Jiang, M., Lupidi, A. M., 等。用于机器学习的人工智能研究智能体：MLE-bench 中的搜索、探索与泛化。载于第三十九届神经信息处理系统年会，2025年。网址https://openreview.net/forum?id=RwfrdKSgCE。Trirat, P., Jeong, W., 和 Hwang, S. J. AutoML-agent：用于全流水线自动化机器学习的多智能体大语言模型框架。载于第四十二届国际机器学习会议，2025年。网址https://openreview.net/forum?id=p1UBWkOvZm。Tschalzev, A., Marton, S., Lüdtke, S., Bartelt, C., 和 Stuckenschmidt, H. 从以数据为中心的视角评估表格数据的机器学习模型。载于第三十八届神经信息处理系统年会数据集与基准赛道，2024年。网址 https://openreview.net/forum?id=kWTvdSSH5W。van Breugel, B. 和 van der Schaar, M. 观点：为什么表格基础模型应成为研究重点。载于第四十一届国际机器学习会议，ICML'24，2024年。网址 https://openreview.net/forum?id=amRSBdZlw9。Wu, Y., Chen, K., Zhang, T., Hui, Y., Berg-Kirkpatrick, T., 和 Dubnov, S. 通过特征融合与关键词到描述的增强，进行大规模对比式语言-音频预训练。载于*ICASSP 2023 - 2023*年*IEEE*国际声学、语音与信号处理会议（*ICASSP*），第1–5页，2023年。doi：10.1109/ICASSP49357.2023.10095969。网址 https://doi.org/10.1109/ICASSP49357.2023.10095969。Xin, D., Miao, H., Parameswaran, A., 和 Polyzotis, N. 生产环境机器学习流水线：实证分析与优化机会。载于*2021*年国际数据管理会议论文集，SIGMOD '21，第2639–2652页，2021年。ISBN 9781450383431。doi：10.1145/3448016.3457566。网址 https://doi.org/10.1145/3448016.3457566。

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., 和 Chen, X. 大语言模型作为优化器。载于第十二届国际学习表征会议，2024a。网址 https://openreview.net/forum?id=Bb4VGOWELI。

Yang、Jimenez、Wettig、Lieret、Yao、Narasimhan 和 Press。Swe-agent：智能体计算机接口支持自动化软件工程。神经信息处理系统进展，37：50528–50652，2024b。网址 https://openreview.net/forum?id=mXpq6ut8J3。

Yang, X., Yang, X., Fang, S., Xian, B., Li, Y., Wang, J., Xu, M., Pan, H., Hong, X., Liu, W., 等。R&D-Agent：通过大语言模型驱动的自动化研究、开发和演进，实现数据驱动的人工智能解决方案构建自动化。*arXiv* 预印本 *arXiv:2505.14738*，2025年。网址 https://arxiv.org/abs/2505.14738。Yu, G.-I., Amizadeh, S., Kim, S., Pagnoni, A., Zhang, C., Chun, B.-G., Weimer, M., 和 Interlandi, M. Windtunnel：迈向超越单一模型的可微机器学习流水线。*VLDB*基金会会刊，15(1):11–20，2021年9月。ISSN 2150-8097。doi: 10.14778/3485450.3485452。网址 https://doi.org/10.14778/3485450.3485452。Zhang, S., Xu, Y., Usuyama, N., Xu, H., Bagga, J., Tinn, R., Preston, S., Rao, R., Wei, M., Valluri, N., Wong, C., Tupini, A., Wang, Y., Mazzola, M., Shukla, S., Li- den, L., Gao, J., Crabtree, A., Piening, B., Bifulco, C., Lungren, M. P., Naumann, T., Wang, S., 和 Poon, H. 基于一千五百万个图像-文本对训练的多模态生物医学基础模型。*NEJMAI*，2(1)，2024。doi: 10.1056/AIoa2400640。网址 https://ai.nejm.org/doi/full/10.1056/AIoa2400640。

A. 附录

A.1 智能体流水线生成器的不足 .14A.2 相关工作 .15A.3 Skrub DataOps 流水线 .16A.4 语义数据算子的详细示例 .16A.5 语义数据算子的优化细节 .25A.6 第 5.1 节的详细说明：“语义数据算子改进并简化表格机器学习流水线” . . . . . 25A.7 第 5.2 节的详细设置：“语义数据算子的表现不逊于或优于专用方法”.55A.8 第 5.3 节的详细说明：“不同大语言模型在语义数据算子代码合成中的优化效果” .56

A.1. 智能体流水线生成器的不足

我们考察了智能体流水线生成器的不足，并分析了用于评估其机器学习工程能力的基准测试中存在的问题。我们的一些见解源于在主要实验中使用 AIDE（Jiang 等人，2025a）和 mini-SWE-agent（Yang 等人，2024b）的经验，详见 A.6.2 节。

由于智能体轨迹不可获取而导致的不透明性。由于许多已发表的基准测试既不向研究社区提供其生成的流水线代码，也不提供生成的智能体轨迹，尽管研究人员提出了明确请求，因此很难判断近期论文中实际的基准测试结果。例如，MLEBench（Chan 等人，2025年）拒绝提供轨迹（https://github.com/openai/mle-bench/issues/28），而截至1月22日，AIRA-dojo（Toledo 等人，2025年）尚未回应提供轨迹的请求（https://github.com/facebookresearch/aira-dojo/issues/12）。

不可运行流水线的比例很高。根据我们使用 AIDE 的经验，相当大一部分生成的流水线代码初始时无法执行，在多达四分之一的情况下都是如此。此外，我们发现有若干情况下，AIDE 尝试使用未安装的库，导致生成的所有代码都无法运行。这需要重启智能体，并手动更新依赖项，或使用专门的提示词鼓励其改用其他库。例如，在尝试将专门的单细胞数据整合和细胞类型预测任务纳入我们的基准测试时，我们观察到 AIDE 和 mini SWE-agent 都多次无法读取输入数据。在 第 5.1 节 的 SIGMOD 编程竞赛任务中，智能体需要大量提示——包括明确的需求、约束和禁止事项——以及一个基线方案，才能生成可执行的流水线。尽管有这些指导，生成的流水线仍未达到运行时间要求，性能也不如给定的基线方案。mini SWE-agent 在 SIGMOD 编程竞赛中的召回率为 0。001 。最后，如果某项任务未出现在 表 2 中，则 AIDE 或 mini SWE-agent 生成的流水线要么无法运行，要么结果不佳。

生成的流水线中的数据泄漏问题。根据我们的经验，多个智能体生成的流水线都存在数据泄漏问题。我们经常发现一种模式：将训练数据和测试数据拼接起来后进行联合预处理（例如缩放或插补）。这种做法在 Kaggle 代码中很常见，因为其目标并非训练出具有泛化能力的型号，而是在预先定义的测试集上取得良好表现。我们推测，大语言模型学会并复现了这种错误模式。由于大多数基准测试都不会公开最终的流水线代码（如上文所述），因此很难确认这一数据泄漏问题是否影响了当前的基准测试结果。我们能找到的唯一可用生成代码来自 CatDB 论文（Fathollahzadeh 等人，2025年），其补充仓库包含使用其系统和 AIDE（Jiang 等人，2025a）生成的流水线。AIDE 最近赢得了 MLEBench（Chan 等人，2025年）基准测试。初步（非穷尽）调查显示，这两个系统都倾向于复现 Kaggle 中的数据泄漏问题。我们为每个系统提供三个代码位置作为证据：

AIDE 流水线中的数据泄漏示例。

- https://github.com/CoDS-GCS/CatDB/blob/main/Examples/AIDE-baseline/Airline/iteration-1-RUN.py#L18
- https://github.com/CoDS-GCS/CatDB/blob/main/Examples/AIDE-baseline/Bike-Sharing/iteration-1-RUN.py#L14
- https://github.com/CoDS-GCS/CatDB/blob/main/Examples/AIDE-基线/房屋-Sales/iteration-3-RUN.py#L33

CatDB 流水线中的数据泄漏示例 .

- https://github.com/CoDS-GCS/CatDB/blob/main/Examples/CatDB/Airline/iteration-2-RUN.py#L31
- https://github.com/CoDS-GCS/CatDB/blob/main/Examples/CatDB/房屋-Sales/teration-2-RUN.py#L37
- https://github.com/CoDS-GCS/CatDB/blob/main/Examples/CatDB/Financial/iteration-2-RUN.py#L56

数据污染对 Kaggle 任务的潜在影响。近期智能体机器学习工程基准的另一项隐忧是，它们依赖 Kaggle 任务，而这些任务很可能已出现在受评商业模型的预训练语料库中。尽管难以直接验证这种污染，但近期竞赛中观察到的性能下降强烈暗示了污染的存在。例如，在 MLE-bench（Chan 等人，2025年）中，gpt-4o 在 2022年之后启动的竞赛中不再能获得奖牌（见 Chan 等人2025年的图 9）。这一变化与其预训练数据在 2023年末截止相吻合（日期引自 https://openai.com/index/gpt-4o-system-card/）。

对表格预测任务的狭隘聚焦。另一个开放性问题是，当前的MLE智能体在Kaggle式预测任务之外表现如何。它们的提示词明确地将它们限定于这类狭隘任务，参见 https://github.com/WecoAI/aideml/blob/main/aide/agent.py#L178 以了解AIDE，以及 https://github.com/facebookresearch/aira-dojo/blob/main/src/dojo/config_dataclasses/task/mlebench.py#L14 以了解AIRA-dojo在MLEBench上的情况。

A.2. 相关工作

正文仅包含相关工作的缩略版本，我们在此提供完整版本。我们详细讨论表格与智能体机器学习流水线及其局限性。

表格机器学习流水线。表格机器学习流水线通常使用 pandas 数据框（pandas 开发团队，2020）来准备、整合和清洗数据，然后组合来自 scikit-learn（Pedregosa等人，2011）的所谓“估计器”（scikit-learn，2025），以编码特征、训练模型并执行超参数搜索。此类流水线在实践中被广泛采用（Psallidas等人，2022）；截至2025年10月，根据 pypistats.org，scikit-learn 和 pandas 软件包的月下载量合计超过6亿次。此外，scikit-learn 的估计器抽象已被其他流行的机器学习库采用，包括 SparkML（Meng等人，2016）和 TensorFlow Transform（Baylor等人，2017），并构成了机器学习流水线持续研究的基础（Karlaš 等人，2024；Petersohn 等人，2020；Grafberger 等人，2023）。尽管其人气很高，当前的表格机器学习流水线生态系统仍有若干重大局限。最值得注意的是，数据框和数据操作在 scikit-learn 的估计器抽象中并非一等公民；该抽象旨在对行数固定的单张表应用线性变换序列（skrub，2025b）。因此，诸如筛选、连接、聚合以及处理多个输入表等常见操作均不受支持。这一局限性才刚刚开始由 skrub 库中引入的“DataOps”抽象（第 A.3 节）加以解决（skrub，2025a）。集成外部专家库（例如用于自动化特征工程）仍然困难（Schäfer 等，2025）。此外，数据操作通常被排除在端到端优化方法之外（Yu等人，2021），因为它们不可微。

智能体式机器学习流水线生成。近期研究探索了利用由自然语言任务描述驱动的智能体式方法，全面自动化地端到端生成表格预测流水线（Toledo 等人，2025年；Aygün 等人，2025年；Fathollahzadeh 等人，2025年；Nam 等人，2025a；Fang 等人，2025年；Yang 等人，2025年；Nam 等人，2025b）。A.1 节讨论了此类智能体式方法的局限性。我们认为，语义数据算子为智能体提供了一种互补的抽象：可复用的基本构件，有望使智能体将对高级流水线结构的推理，与对单个数据操作的综合与优化分离开来。

基于语义数据算子的声明式编程。将大语言模型驱动的语义数据算子集成到 SQL 中以查询多模态数据的声明式编程，是数据库领域的一个活跃研究方向（Patel 等人，2025年；Jo 与 Trummer，2024；Liu 等，2025年；Lao 等，2025年；Russo 与 Kraska，2026）。主要云服务供应商近期已将语义算子集成到生产级数据库系统中（谷歌，2025年；Databricks，2025年；Liskowski 等，2025）。对于语言模型编程，DSPy（Khattab 等，2024）提供了一个广泛使用的声明式框架，可根据底层数据和模型自动优化程序提示（Opsahl-Ong 等，2024）。

通过提示进行程序优化。提升机器可判分程序的一种新近方法是“提示优化”（Yang 等人，2024a），即通过反馈和下游

性能得分，迭代调用代码合成大语言模型，以改进其生成的代码。这一范式已应用于多种任务，包括表格预测的自动化特征工程（Hollmann 等人，2023b）、语言模型编程和复合人工智能系统的提示优化（Opsahl-Ong 等人，2024；Agrawal 等人，2025年；Lee 等人，2025年），以及科学领域的算法发现（Aygün 等人，2025年）。机器学习流水线非常适合这一框架，因为它们在留出数据上的性能提供了一个自然的优化目标。

A.3. Skrub DataOps 流水线

skrub 库近期推出了 DataOps 流水线（skrub，2025a），这是一种轻量级抽象，用于构建基于数据框的多表机器学习流水线；它将流水线表示为计算图，其中节点代表数据操作，边则编码数据流。与标准的 scikit-learn 流水线不同，后者仅限于对单个表格执行一系列线性变换，且行数固定；DataOps 抽象涵盖了跨多个表格的复杂数据操作。重要的是，它将整个流水线——包括多表数据框操作——整合为统一的 scikit-learn 风格预测器，并提供 Fit 方法以进行端到端训练，以及预测方法，用于对未见过的数据进行推理。

如需了解更多详情和代码示例，请参阅 skrub 项目的在线文档和近期演讲：

- 使用 Data Ops 构建复杂的多表流水线

https://skrub-data.org/stable/data_ops.html
- Skrub：面向 DataFrame 的机器学习，PyData Paris

2025https://youtu.be/k9MNMDpgdAk?si=mw-HjEw-x5M0rvwE&t=694
- 数据科学中的整洁代码——Skrub DataOps，Dot AI

2025https://youtu.be/bQS46Pgbnvc?si=wqduC_fASILSO0Ql&t=769

A.4. 语义数据算子的详细示例

我们将详细讨论三个语义数据算子作为示例。我们选择了复杂且在实验中居于核心地位的算子。所有语义算子的完整实现均可在https://github.com/deem-data/sempipes/tree/v1/sempipes/operators获取。

使用 sem_agg_features 进行聚合特征生成。sem_agg_features(D, D_{agg}, c_{join}, s, k) 操作会依据自然语言指令 s，对第二个数据框 D_{agg} 进行聚合，在数据框 D 中创建k 新的特征列，并通过联接将聚合结果添加到 D 。用关系代数形式化后，SEMPIPES 需要合成一个函数 $o_{\text{synth}}(D, D_{\text{agg}}) := D \bowtie_{c_{\text{join}}} \gamma_{c_{\text{join}};A_1,\ldots,A_k}(D_{\text{agg}})$，该函数对 D应用 k 个聚合A_1，……，A_k，并按列 c_{join}分组，随后在 D 与聚合结果之间执行左外连接，连接条件为 c_{join} 。为了帮助大语言模型生成代码，SEMPIPES 在训练时构建丰富的上下文：计算输入数据框的特征，例如列名、列类型、描述性统计和数据样本。此外，它还指定一个代码骨架 sem_agg_join(left_join_column, left_df, right_join_column, right_df) -> DataFrame，并为其提供强类型签名；同时检查整个流水线代码的计算图，以确定机器学习任务的类型（例如分类或回归）、所训练型号的具体类型，以及输入数据和标签的自然语言描述（如果代码中对此进行了注释）。我们采用思维链提示，要求大语言模型在每个聚合函数前添加代码注释，描述生成的特征、解释选择该特征的理由，并说明该特征如何为下游模型提供有用的现实世界知识。SEMPIPES 提供多种方法来验证和引导代码生成。对于大语言模型给出的候选函数，SEMPIPES 会从输入数据框 D 中抽取 100 行，并筛选 D_{agg}，使其与 D的样本中 90% 的连接键相匹配，从而构造用于验证的输入样本。随后在这些输入样本上运行候选代码，并检查结果是否仍包含原有行数，以及是否保留了 D 中的原始列。如果发生任何运行时错误或验证错误，系统会将错误消息反馈给大语言模型，并通过重试机制生成改进版候选代码。SEMPIPES 还允许

示例。我们提供了一个具体示例，展示 SEMPIPES 为一个从购物篮聚合产品数据的欺诈检测流水线合成的代码。

包含语义算子的流水线。

```
1  def pipeline() -> skrub.DataOp:
```

```
产品 =skrub.var("产品")
购物篮 =skrub.var("数据")
标签 = skrub.var("标签")

购物篮 =baskets.skb.mark_as_X().skb.set_description("潜在 欺诈性购物 购物篮的 产品")
标签 = 标签.skb.mark_as_y().skb.set_description(
    "标记 表示 是否 the 购物篮 is 欺诈的 (1) 或不是 (0)"
)

聚合的 =购物篮.sem_agg_features(
    产品 ,
    左侧匹配键="ID",
    右侧匹配键="basket_ID",
    nl_prompt="""
        生成 产品 特征 那些是 表明 潜在地 欺诈性购物篮, make 它容易 to 区分
        异常的 购物篮 与常规的区分开来! 可能 有助于结合 不同的 产品统计数据。
        """,
    名称="购物篮_聚合_特征",
    多少=1,
)

编码 = 聚合.skb.应用(TableVectorizer())
返回 编码.skb.应用(直方图梯度提升分类器(随机种子=0), y=标签)
```

对应的合成算子代码。

```
导入 pandas 作为pd
导入 NumPy 作为np

def _sem_agg_join(左连接列, 左侧数据框, 右连接列, 右侧数据框):
    """
    汇总 right_df 的特征，并通过左连接将其合并到左侧数据框。

    参数:
        左连接列（字符串）: 该列 in 左侧数据框用于连接的列。
        左侧数据框（pd.DataFrame）: 该 左侧数据框。
        right_join_column（字符串）: 列 在右侧数据框中用于连接。
        right_df（pd.DataFrame）: 要聚合并连接的右侧数据框。

    返回:
        pd.DataFrame: 连接了新聚合特征的左侧数据框。
    """

    # 在以下级别聚合所需特征: 购物篮 ID
    agg_df = 右侧数据框.groupby(右连接列).agg(
        购物篮现金价格总额=('现金价格', '求和'),
        total_basket_product_units=('购买商品数量', '求和'),
        # 特征: 购物篮中不同商品的数量
        # 描述: This 特征统计不同项目的数量 '商品' 类别 已购买 在每个购物篮中。
        # 理由: 异常或 欺诈性购物篮可能经常 呈现 异常的商品 多样性。 非常多的 high 数量
        # of 独特 商品可能表明 不加选择的 “砸抢式” 试图 购买各种 低值
        # 项目, or 欺诈者 测试 多种产品类型中的被盗银行卡额度 。 反之, 极其
        # 少量独特 商品（例如, a 购物篮中有 仅有一个非常 昂贵且独特的 可能会被转售的商品
        # 转售 很快）可能 also 是一个危险信号。 This 特征 提供购物篮复杂度的定量指标,
        # 其 不同于总内存 货币 价值或数量 单位。
        # 现实世界知识: 这项 特征 有助于捕捉 行为的 欺诈者的模式, 他们可能
        # 任一 广泛地 获取一系列不同的 产品（多样化商品 商品）, 或有针对性地 锁定特定的
        # 商品类型（低 多样性）. 这 通常 偏离 from 典型的 客户 购买 行为, 从而
        # 有助于 进行异常 检测。
        购物篮中不同商品的数量=('商品', '唯一值数量'),
        # 特征: 购物篮中商品现金价格最大值
        # 描述: This 特征捕获 最高现金 任意产品的价格 单个 产品条目 在一个 给定 购物篮,
        # 考虑 the '现金价格' 列, 该列 表示乘以 之前的单位价格 '购买商品数量'。
        # 理由: 欺诈 活动可能 往往 针对高价值 项目 以便快速 转售（例如, 电子产品, 昂贵的
    家具）。
        # By 隔离 the 价格最高的 单个 购物篮中的商品 购物篮, 我们可以 flag 包含以下内容的购物篮 比例失衡地
        # 昂贵 产品可能 成为目标 用于信用卡欺诈 或退货 欺诈。 相反, 由.....组成的购物篮
        # 仅包含极低 价格商品也可能 并不常见 并表明存在试图利用最低 购买 限额的行为。
        # 现实世界知识: 在现实世界的欺诈 场景中, 攻击者通常 优先处理 高 转售价值的商品
        # 以最大化 非法 收益。 This 特征 直接 捕获 the “最高 风险” 交易中的商品
        # 从价值 视角, 提供 洞见 不同于购物篮总价值或平均 商品价值。
        购物篮中商品现金价格最大值=('现金价格', 'max'),
        # 特征: 购物篮中的服务履约商品数量
        # 描述: This 特征统计 数量 购物篮中的商品 被归类为 服务或 履约。
        # 理由: 服务费和履约费 通常 具有现金 价格为 0 或 固定的较低值。 它们的
        # 存在与 数量 in a 购物篮可以揭示 具体 购买模式。 大量 此类商品的数量可能
        # 异常, 可能 表明 试图 绕过最低 购买要求 用于某些 促销活动或填充
        # 笔交易 用于其他 欺诈 目的（例如, 创建 fake 交易量 用于银行卡测试或掩盖
        # 更少的高价值 实物与 众多 低成本/无成本 “填充内容” 项目）。 This 特征 区分名义物品
        # 与实体 产品。
        # 现实世界知识: 一些 欺诈 方案 将名义上的商品或可用内存作为更大策略的一部分加以利用。
        # 例如, 数量过多的 服务可用于使交易看起来 合法, 而
        # 高价值 实物 商品是 正在购买 with 被盗凭证, 或用于测试 信用 card 有效性 with 最小的
        # 金融 风险。 This 直接 针对这类行为的 异常。
        购物篮中的服务履约商品数量=('是否为服务或履约', '求和'),
        # 特征: 购物篮现金价格标准差
        # 描述: This 特征计算 的标准差 `现金价格` 针对某一特定范围内的所有商品 购物篮。
        # 理由: 一个非常 high 标准差 表明一个 购物篮中包含 多种商品 价格（例如, 组合
        # 非常 便宜 填充 商品与 一个昂贵的 目标商品）, 这 can 可能是一个异常的 常见的购买模式
        # in 欺诈 活动。 反之, 非常 极低或为零的标准差（所有商品价格相近）可能
```

```
        # 也可能是特定的（例如，批量采购相同的高价值商品用于转售，或试图购买许多# 廉价商品）。此特征用于捕捉一笔交易中商品价值的离散程度或异质
                                                                                                                                                e
        # 购买行为，而另一些人则专门购买价格一致的昂贵商品。标准差有助于识别这些结构性                                                                                                                                l
        # 购物篮价值构成中偏离顾客典型购买习惯的异常。
        购物篮现金价格标准差'),# 特征：购物篮中的不同品牌数量 # 描述：此特征统计每个购物篮中不同'品牌'(制造商/品牌)类型的数量。# 理由：与商品多样性


    唯一
        # 品牌（例如，从单一特定制造商（如'苹果'或'零售商'）处购买大量商品）可能意味着# 针对性地采购易于转售的特定品牌商品，这通常与欺诈有关。反之，购


        # 特征：最小值 _item_cash_价格 _in_购物篮
          # 描述：此特征计算给定购物篮中任意产品系列的最低现金价格。# 理由：此特征与“购物篮中商品现金价格最大值”相辅相成，可更全面地呈现购物篮中的价格范围。#


    ).重置 _索引()

    # 特征：basket_item_value_per_product_unit # 描述：此特征计算购物篮中每个单独产品单位所贡献的平均现金价值。# 原理：欺诈购物篮可能会在每件商品价值方面呈现


    agg_df.loc[agg _df['总内存_购物篮 _产品 _单位 '] == 0, '购物篮 _item_值 _per_产品 _unit'] = 0

    # 特征：购物篮价格最大集中度 # 描述：此特征计算单件商品的最高现金价格占整个购物篮现金价格总额的比率。# 它衡量购物篮总价值中有多少集中在价格最高的商品上。#
```

```
139 # 删除用于计算的中间求和列，因为它们不是最终所需的特征 140 agg_df = agg_df.drop(columns=['购物篮现金价格总额', '购物篮商品总单
位数
{rig
how=
```

使用 `sem_extract_features`进行特征提取。sem_extract_features(D, C_{out}, s) 操作按照相应的自然语言指令 s，从 DataFrame D 中提取特征列 C_{out}。为该算子合成代码时，SEMPIPES 首先确定 D 所有输入列的模态（文本、图像、音频），并提供带类型签名的代码骨架 sem_extract_features(df: DataFrame, features_to_extract: list[dict[str, object]]) ->DataFrame。接下来，系统提示大语言模型为每个输出列生成定制提取代码 c（使用用户提供的指令 s），涵盖所有特征列（c, s）$\in$（C_{out}, S_{out}）。提示要求大语言模型在零样本模式下，使用 HuggingFace 的 Transformers 库中的合适预训练模型合成提取代码，选择负载最低的可用图形处理器，使用批处理以高效利用图形处理器，并向用户显示进度条。SEMPIPES 会在 50 个输入元组的样本上验证候选代码，并检查是否已添加所请求的特征列。如果出现时长或验证错误，系统会将错误消息反馈给大语言模型，并应用重试机制来生成改进后的候选代码。

示例。我们提供一个具体示例，展示 SEMPIPES 为推文分类流水线中的特征提取所合成的代码，其中 SEMPIPES 针对情感检测、毒性检测和命名实体识别选择不同型号。

包含语义算子的流水线。

```
def pipeline() -> skrub.DataOp:
    推文 =skrub.var("数据")
    标签 = skrub.var("标签")

    推文 = 推文.skb.mark_as_X()
    标签 = 标签.skb.mark_as_y()

    输出列 ={
        "可能有毒": "响应 with '1' 如果一条推文具有毒性，并且 '0' 如果它的 文本不具有毒性。",
        "情感标签": "分类 整体的 情感 的 推文作为 '正面', '负面', or '中性.",
    }

    已标注的推文 =推文。语义_提取_特征（
        自然语言提示="提取 特征 从文本推文中 有助于预测毒性的。",
        输入列=["文本"] ,
        名称="推文特征" ,
        输出列=输出列 ,
        通过代码生成=True ,
        将代码打印到控制台=True ,
    )

    编码后的推文 =labeled_tweets.skb.应用(TableVectorizer())
    返回 encoded_tweets.skb.应用(随机森林分类器(random_state=0), y=标签)
```

对应的合成算子代码。

```
导入 PyTorch
导入 pandas 作为pd
从tqdm.auto 导入 tqdm
从 Transformers 导入（
    自动分词器 ,
    自动序列分类模型 ,
    流水线 ,
）

def pick_gpu_by_free_mem_torch():
    断言 torch.cuda.is_available(), "No CUDA 设备"
    free = []
    对于i在范围(torch.cuda.device_count()):
        # 返回（可用内存, 总内存）in 字节 对于该设备
        f, t = PyTorch.cuda.mem_get_info(i)
        free.append((f, i))
    free.sort(reverse=True)
    _, idx = free[0]
    打印(f"已选择 图形处理器: {idx}")
    返回 idx

图形处理器_idx = pick_gpu_by_可用内存_mem_PyTorch()
设备 = torch.device(f"cuda:{gpu_idx}")# 使用所选图形处理器进行模型推理
```

```
待提取特征 = [{'特征名称': '情感标签', '特征提示':"对整体情感进行分类
    '中性.", '输入列': ['文本']},{'特征名称': '可能有毒','feature_prompt':"如果推文含有毒性，则回答'1' ，如果推文文本不含毒性，则回答
    输入列': ['文本']},{'特征名称': '脏话','feature_prompt': '以逗号分隔的文本中所有脏话列表
['文本']}]定义 提取特征(df: pd.DataFrame , 待提取特征: 列表[字典[字符串, 对象]]) -> pd.DataF


 # 情感分析流水线（零样本，稳健） sentiment_model_name = "cardiffnlp/twitter -roberta -base -
```

推文作为 '正面', '负面', or '0' ", ' 词语' , '输入列':

rame:

```
114 词元  = re.findall(r"\b\w+\b", text.lower()) 115 found  = [w for w in tokens if w in swear_words_set]
116 
batc
121
122
wea
swea
```

使用 `sem_gen_features`进行特征生成。`sem_gen_features`$(D,\ k,\ s)$ 算子可根据自然语言指令 k 在 DataFrame 中生成 D 额外的特征列 s。我们的实现基于 CAAFE（Hollmann 等人，2023年b）方法所使用的提示词，该方法以开放许可证发布于 https://github.com/noahho/CAAFE。主要区别在于，SEMPIPES 在代码合成过程中会加入更多关于当前流水线的上下文信息，例如检查流水线的计算有向无环图，以确定所用模型的类型、当前问题（例如回归或分类）、目标列的描述，并获取该算子的确切输入数据（而不仅仅是流水线的输入数据集，因为它可能已被其他操作修改）。

示例。我们提供一个具体示例，展示 SEMPIPES 如何为一个经过重写、用于预测电影票房收入的 Kaggle 流水线合成特征生成代码。

包含语义算子的流水线。

```
def sempipes_pipeline():
    数据 = skrub.var("数据")
    收入 =data.revenue
    数据 = 数据.drop(columns=["收入"])

    电影统计数据 =data.skb.mark_as_X()
    电影统计数据 = movie_stats.skb.set_description("""
    在一个 电影创造了约 417 亿美元收入的 in 2018年, the film 行业 更多的是 更受欢迎。 但是哪些电影
      的票房最高? 导演有多重要? 预算呢? 对于一些电影, it's “你说
      你好。 对于其他电影, 预告片 未达到预期, 并且 你认为 “我们 have 这里有一个 沟通失败。” In
       这场竞赛, 你 展示了 with 数千部.....的元数据 过往电影 来自《 电影》 数据库, 尝试 预测
      它们的总 全球 票房收入。 Data 分数 所提供的包括演员阵容, 剧组, 剧情关键词, 预算, 海报, 发行
      日期, 语言, 制片公司, and 国家。 你可以收集 其他 公开 可用 data 使用 in 您的型号
      预测结果, 但本着本次竞赛的精神, 仅使用 data 当时本应 已可获得的信息 在电影上映前。 It
      你的任务是 预测 国际票房收入 对于每部 电影。 对于测试集中的每个标识符, 你必须 预测
      收入变量的值。 提交结果将根据 均方根对数误差（RMSLE）进行评估 在 预测值
      与实际值之间 收入。 为避免票房收入高的电影占据过大权重, 使用对数处理。
    """)

    收入 =收入.skb.mark_as_y()
    收入 =收入.skb.set_description("the 国际票房收入 一部电影")

    电影统计数据 =电影统计数据.sem_gen_features(
        nl_prompt="""
            创建有助于预测以下内容的附加特征 一部电影的票房收入。
            考虑方面 like 类型, 制作 细节, 演员阵容, 剧组, and any 其他相关 信息
            可能影响一部电影的 财务 成功。 其中一些属性采用 JSON 格式 ,
            因此, 你可能需要解析它们以提取有用信息。
        """,
        名称="附加电影特征" ,
        数量是多少=25,
    )

    JSON列 = [
        "属于集合",
        "类型",
        "制作公司" ,
        "制作国家" ,
        "口语语言",
        "关键词",
        "演员阵容" ,
        "剧组" ,
    ]

    定义 列名清理(df):
        df.columns =df.columns.字符串.replace(r"[^A-Za-z0-9_]+", "_", 正则表达式=True)
        返回 df

    电影统计数据 =电影统计数据.drop(columns=JSON列)

    编码器 =skrub.TableVectorizer()

    X = 电影统计数据.skb.应用(编码器)
    X =X.fillna(-999)

    X =X.skb.应用函数(列名清理)

```

```
    y_log = 收入.skb.应用函数(np.log1p)

    参数 ={
        "目标函数": "L1 回归",
        "指标": "均方根误差",
        "估计器数量": 3000,
        "学习率": 0.003,
        "特征比例": 0.7,
        "装袋采样比例": 0.7,
        "装袋频率": 1,
        "L1正则系数": 0.1,
        "L2正则系数": 0.1,
        "叶子节点数": 31,
        "详细输出": -1,
        "作业数": -1,
        "随机种子": 42,
        "提升类型": "梯度提升决
    }
    型号 = lgb.LGBMRegressor(**params)
    预测结果 = X.skb.应用(型号, y=y_log)

    定义 变换时取指数(outputs, 模式=skrub.eval_mode()):
        if mode 在 {"变换", "预测"}:
            返回 NumPy 指数减一函数(outputs)
        返回 outputs

    返回预测结果.skb.应用函数(变换时取指数)
```

对应的合成算子代码。

```
导入 JSON
导入 ast
导入 NumPy 作为np

# (特征: 包含集合 - 二元 指示器 如果电影 属于 一个集合)
# 输入 样本: '属于集合': [nan, nan, nan]
df['包含集合'] = df['属于集合'].应用(lambda x: 1 if isinstance(x, 字符串) 和长度(x) > 2 否则 0)

# (辅助 函数 用于安全 JSON 解析)
def 安全字面量求值(x):
    如果 isinstance(x, str) and x.strip():
        try:
            返回 ast.literal_eval(x)
        除外 (ValueError, SyntaxError):
            返回 []
    返回 []

# (特征: 类型数量 - 数量 相关类型 的电影)
# 输入 样本: '类型': ["[{'标识符': 18, '名称': '剧情 '}]", "[{'标识符': 18, '名称': '剧情 '}, {'标识符': 53, '名称': '惊悚 '}]", "[{'标识符': 10751, '名称': '家庭 '}, {'标识符': 16, '名称': '动画 '}, {'id': 35, '名称': '喜剧 '}, {'id': 12, '名称': '冒险 '}]"]
df['解析后的类型'] = df['类型'].apply(安全字面量求值)
df['类型数量'] = df['解析后的类型'].apply(len)

# (特征: 类型为剧情 - 二元 指示器 for 「剧情」 类型)
# (特征: 类型为喜剧 - 二元 指示器, 用于 '喜剧' 类型)
# (特征: 类型_is_惊悚 - 二元 指示器 for '惊悚' 类型)
# (特征: 类型_is_动作片 - 二元 指示器 for '动作片' 类型)
# 输入 样本: '类型': ["[{'标识符': 18, '名称': '剧情 '}]", "[{'标识符': 18, '名称': '剧情 '}, {'标识符': 53, '名称': '惊悚 '}]", "[{'标识符': 10751, '名称': '家庭 '}, {'标识符': 16, '名称': '动画 '}, {'标识符': 35, '名称': '喜剧 '}, {'标识符': 12, '名称': '冒险 '}]"]
df['类型为剧情'] = df['解析后的类型'].apply(lambda x:1 if {'标识符': 18, '名称': '剧情'}位于x 否则0)
df['类型为喜剧'] = df['解析后的类型'].应用(lambda x:1 如果{'标识符': 35, '名称': '喜剧'}位于x 否则0)
df['genre_is_Thriller'] = df['解析后的类型'].应用(lambda x:1 如果{'标识符': 53, '名称': '惊悚'}在x 否则 0)
df['genre_is_Action'] = df['解析后的类型'].应用(lambda x:1 如果{'标识符': 28, '名称': '动作片'}在x 否则 0)
删除 df['解析后的类型']

# (特征: num_production_companies - 数量 制片公司)
# 输入 样本: '制作公司': ["[..." , "{'名称': 'DiNovi Pictures' , '标识符': 813}]" , "[...]"]
df['parsed_production_companies'] = df['production_companies'].应用(safe_literal_eval)
df['num_production_companies'] = df['parsed_production_companies'].应用(长度)

# (特征: has_WarnerBros - 二元 华纳兄弟指示器 is a 制作公司)
# 输入样本: '制作公司': ["[{'名称': '二十世纪 Fox Film 公司', '标识符': 306}, {'名称': 'Regency Enterprises' , '标识符': 508}, {'名称': 'Fox 2000 Pictures' , 'id': 711}, {'名称': '金牛座 电影' , '标识符': 20555} , {'名称': 'Linson 电影' , '标识符': 54050} , {'名称': 'Atman Entertainment' , '标识符': 54051} , {'名称': 'Knickerbocker 电影' , 'id': 54052}]" , "[{'名称': 'DiNovi Pictures' , '标识符': 813}]" , "[{'名称': '阿德曼 动画公司' , '标识符': 297}, {'名称': 'Studio Canal' , '标识符': 5870} , {'name': 'Anton Capital Entertainment (ACE)', 'id': 20664}]"]
df['has_WarnerBros'] = df['parsed_production_companies'].应用(lambda x:1 如果存在(comp['名称'] == '华纳兄弟' 对于 comp 在x) 否则 0)

# (特征: 制片国家数量 - 数量 制片国家)
# 输入 样本: '制作国家': ["[{'ISO 3166-1': '德国', '名称': '德国 '}, {'ISO 3166-1': '美国', '名称': '美利坚合众国 美洲 '}]" , "[{'ISO 3166-1': 'US', '名称': '美利坚合众国 '}]" , "[{'ISO 3166-1': 'FR', '名称': '法国 '} , {'ISO 3166-1': 'GB', '名称': '英国 '}]"]
df['解析后的制作国家'] = df['制作国家'].应用(安全字面量求值)
df['制片国家数量'] = df['解析后的制作国家'].应用(长度)

# (特征: 是否为美国制片 - 二元 指示器, 表示美国是否为 生产国)
# 输入 样本: '制作国家': ["[{'ISO 3166-1': '德国', '名称': '德国 '}, {'ISO 3166-1': '美国', '名称': '美国的 美洲 '}]" , "[{'ISO 3166-1': 'US', '名称': '美利坚合众国 '}]" , "[{'ISO 3166-1': 'FR', '名称': '法国 '} , {'ISO 3166-1': 'GB', '名称': '英国 '}]"]
```

```
50 df['是否为美国制片'] = df['解析后的制片国家'].apply(lambda x: 1 if any(country['ISO 3166-1'] == 'US' for country in x) else 0)
51
52
本: '
'英语
语言

61
'en' f
851,
{'id':
{'id':
1565

66
df['解
2））
\'52f

74
\'名称
解析
后的
权平
\'ord
287,
\'52f

81
l.jpg\'] = df['演员阵容'].apply(lambda x: np.mean([c.get('order', 0) for c in x]) if x else 0) 82 df['演员阵容最小顺序'] = df['解析后的演员阵容'].
apply(lambda x: np.min([c.get('order', 0) for c in x]) if x else 0) 83 删除 df['解析后的演员阵容']84 85 #（特征：导演数量——不重复导演的数量）86 #（特征：制片人数量——不重复
制片
'job '

89
twLI
量']
for c
in
{'cre

98
appl
演' a
析,
x.str
(Valu
```

```
120 # 输入样本: '属于集合 ': [nan , nan , nan, "[{'id': 107467, '名称 ': '最佳电影——合集 ', 'poster_path ': None , ' backdrop_path ': None}
121
'标识
零预
数' d
129
130
df '预
'人气
138
139
'http
元指
nan
146
147
向新
炸弹
她平
appl
154
155
肖恩
158
159
标题
162
163
（特
appl
169
170
df.dr
处理
x.str
ast.li
入样
Time
几周
00 0
Time
对数
else
201
202
预算'
```

```
205 #（特征：人气与运行时间比 - 人气与运行时间的比率，并对零时长进行稳健处理）
206
207
长',
若“标
212
则为
'en').
入样
223
224
'ru').
227
228
'es').
231
232
'bud
```

A.5. 语义数据算子的优化详情

我们提供有关优化器的更多详细信息。

• 验证效用的计算– 优化器目前会在外循环变异步骤中，使用用户提供的验证集进行五折交叉验证，以计算验证效用。请注意，用户可以配置评估方法。
• *LLM* 引导的变异方法– EVOLVE 函数的具体实现因算子而异。目前，每个算子都会根据输入数据特征、流水线上下文、记忆轨迹和灵感构建自定义提示，通过大语言模型合成其代码的新版本。同样，记忆轨迹和灵感的结构也因算子而异。我们当前的实现会在记忆轨迹和灵感中利用合成代码的历史记录及其对应的效用分数，其中还包括合成代码的大语言模型在代码注释中提供的文本推理。

• 学习任务和模型类型的推理– 我们的优化器会检查 DataOps 流水线的计算图，以识别最后一个应用 scikit-learn 估计器的 DataOp；该估计器实现了预测器混入类，代表最后应用的模型。接下来，它会使用启发式方法，根据模型类确定学习任务的类型。详情请参阅检查代码，地址为 https://github.com/deem-data/sempipes/blob/v1/sempipes/inspection/pipeline_summary.py#L45。

我们的优化器的完整实现可在以下网址获取：
https://github.com/deem-data/sempipes/blob/v1/sempipes/optimisers/colopro.py#L87。

A.6. 第 5.1 节详情：“语义数据算子改进并简化表格机器学习流水线”

在本小节中，我们介绍选定的任务，展示语义数据算子如何应用于选定的流水线，并说明 SEMPIPES 生成的代码。

A.6.1. 关于所选任务的详细信息

下文将讨论任务选择、数据源和实验方案，参见表 5。

对于每项任务，非优化的 SEMPIPES 流水线最多使用 5 次重试（默认值），优化后的流水线最多使用 24 次试验。若先前生成的代码失败，每次重试都会调用一次大语言模型。根据设计，SEMPIPES 的成本相对较低，因为大语言模型调用次数是固定的，并且随流水线中的语义数据算子数量线性增长。在我们的主要实验中，除特征工程（房价）和实体解析分块（SIGMOD 编程竞赛）任务外，每项任务都包含一个语义数据算子；后两项任务则包含两个语义数据算子。此外，在 SEMPIPES 中，大语言模型仅在训练期间用于代码生成，推理期间不会调用。

表 5：用于评估语义算子的以数据为中心的任务。

以数据为中心的任务	模态	来源
临床信息引导的特征提取	表格	EMNLP'21
实体解析分块	表格 +文本	SIGMOD'22
关系数据上的特征工程	表格	Kaggle
用于模型调试的数据标注	表格 +图像	NeurIPS'23
公平性数据增强	表格	NeurIPS'23

专家编写及智能体生成的流水线代码集合。

临床信息特征提取。该任务围绕用于建模社交媒体文本中心理社会信号的机器学习流水线展开，这些流水线将模型预测结果建立在临床相关症状的基础上。常见做法是设计一种具有层次化模型架构的流水线：先使用弱监督数据训练一组症状分类器，为后续的预测模型计算输入特征（Nguyen 等人，2022年；Lee 等人，2021）。研究发现，这种层次化设计能够提升临床医生对预测结果的可解释性，并增强模型对域外数据的泛化能力。我们从（Lee 等人，2021）（EMNLP'21）的代码库中选取了一个名为 `micromodels` 的流水线。该流水线将人工设计的"micromodels"用于生成输入特征，以便可解释提升模型基于 Reddit 社交媒体对话预测共情。

- 论文：《高效、可解释且可复用系统的微型模型：心理健康案例研究》，EMNLP'21（Lee等，2021）

- 代码仓库：https://github.com/MichiganNLP/micromodels
- 数据：https://github.com/MichiganNLP/micromodels/blob/master/empathy/data/combined_iob.json
- 流水线：`micromodels` - 共情预测流水线的重新实现
- 数据集规模：2,023 个样本
- 指标：用于预测情绪反应水平的最终分类器的 F1分数
- 实验协议：采用 50/50 训练集—测试集划分，并使用五个不同的随机种子
- 实验代码：https://github.com/deem-data/sempipes/tree/v1/experiments/micromodels

实体解析分块。该任务源自 ACM SIGMOD 2022 编程竞赛（Ives 等人，2022年）（一场顶级数据库会议），竞赛要求构建一个用于实体解析（ER）的阻塞系统。实体解析是指在数据集中识别并匹配指向同一现实世界实体的不同元组。对大型数据集进行实体解析时，直接评估所有候选元组对（其数量呈平方级增长）通常会造成难以承受的计算开销。因此，通常会先执行一个预筛选步骤，即阻塞步骤，快速筛除明显不匹配的候选项，并获得一个规模小得多的元组对候选集，随后再通过匹配步骤找出匹配项。竞赛的目标是基于输入数据集的带标签样本，设计一个召回率高效的阻塞函数。我们对竞赛排行榜中的两个高排名基于 Python 的解决方案进行了实验（分别称为 `sustech` 和 `rutgers`），这两个方案应用了大量手动数据清洗和特征提取操作；此外，我们还测试了主办方提供的基线 解决方案。

- 来源：SIGMOD 2022 编程竞赛
- 数据：https://dbgroup.ing.unimore.it/sigmod22contest/task.shtml?content=说明，隐藏测试数据由竞赛组织者提供

- 流水线：`sustech` - SUSTech DBGroup 提交的方案（第 4 名）
- 流水线：`rutgers` - 罗格斯大学提交的方案（第 5 名）
- 流水线：SIGMOD 基线 -竞赛组织者提供的基线
- 数据集规模：训练数据包含 2,006 行；测试数据包含 1,215,129 行；预期输出为 2,000,000 个候选对
- 指标：阻塞函数在隐藏测试数据上的召回率
- 实验协议：仅使用少量训练样本进行代码合成和优化，并在测试数据上进行评估
- 实验代码：https://github.com/deem-data/sempipes/tree/v1/experiments/sigmod

关系数据上的特征工程。我们纳入了来自 Kaggle 的三个表格预测任务的流水线，这些任务需要对关系数据进行大量的特征工程和预处理。我们按照其他基准测试的做法（Chan 等人， 2025年），在 Kaggle 竞赛提供的部分训练数据上评估性能（因为测试集的标签未公开）。我们采用了针对三个 Kaggle 任务的五个公开笔记本，这些笔记本由在竞赛中获奖的高分人类专家设计。

- 来源： TMDB票房预测竞赛 tion
  – 流水线： kaggle-movie-a- 公开笔记本 notebook95778def47– 流水线： kaggle-movie-b - 公开笔记本 特征工程、特征重要性、随机森林(铜牌得主提供)– 数据集规模： 1,002 个样本– 指标： 用于预测收入的均方根对数误差（RMSLE）– 实验协议： 使用五种不同随机种子的 50/50 训练集—测试集划分–实验代码：https://github.com/deem-data/sempipes/tree/v1/experiments/tmdb_票房_预测_预测

- 来源： 房价 - 高级回归技术 竞赛
  – 流水线： kaggle-house-a- 公开笔记本 房价：数据清洗、可视化与建模 (铜牌得主提供)– 流水线：kaggle-house-b - 公开笔记本 房价解决方案：数据清洗–>机器学习 (铜牌得主提供)– 数据集规模： 463 个样本– 指标： 用于预测房价的均方根对数误差（RMSLE）– 实验协议： 五折交叉验证（由于数据集规模较小）–实验代码： https://github.com/deem-data/sempipes/tree/v1/experiments/house_房价_高级_回归_技术

- 来源： Scrabble玩家评分竞赛
  – 流水线： kaggle-scrabble - 公开笔记本 使用XGB回归对Scrabble玩家进行评分 （来自铜牌得主）– 数据集规模： 80,820条玩家评分、72,774场游戏、2,005,499个回合– 指标： 非机器人玩家预测评分的均方根误差（RMSE）– 实验协议： 使用五个不同随机种子的50/50训练集—测试集划分–实验代码：https://github.com/deem-data/sempipes/tree/v1/experiments/scrabble_玩家_评分

用于模型调试的数据标注。我们专注于为模型调试标注图像这一任务，遵循NeurIPS'23上（Chen 等，2023年）提出的方法。该方法使用视觉语言模型，根据预定义的特定领域属性类别对图像进行标注，并利用这些属性发现预测模型表现不佳的数据切片。我们使用论文代码仓库中的hibug 流水线进行实验。该流水线通过视觉问答模型，为数据集上名人照片分类器的验证集计算多个预定义的标注属性。我们比较验证集18,000个样本上八种属性（如发色、年龄等）的标注准确率；对于这些样本，数据集包含真实标签。对于专家代码基线，我们复用了论文代码仓库中的预计算结果。

- 论文："关于人类可解释的模型调试"，NeurIPS'23（Chen 等，2023年）
- 代码仓库：https://github.com/cure-lab/HiBug
- 数据：https://www.kaggle.com/datasets/jessicali9530/celeba-dataset?select=img_align_celeba
- 流水线：hibug - 使用预计算结果，用于属性选择部分，来自CelebA 笔记本
- 数据集规模：18,000 个样本
- 指标：预测属性的平均准确率，并与 CelebA 数据集中的真实标注进行比较
- 实验协议： 八种不同属性的零样本提取
- 实验代码：https://github.com/deem-data/sempipes/tree/v1/experiments/hibug

公平性数据增强。此任务的灵感来自 NeurIPS'23 上Qian 等人（2023年）的研究，该研究构建了一条流水线，用于根据巴西的 SIVEP 数据集预测新冠相关健康结局（Baqui 等人，2021），并使用合成数据增强来提高数据中代表性不足的种族少数群体的 AUROC 分数。由于实验数据现已无法获取，且实验代码安装时遇到问题，我们在公开可用的 SIVEP 数据集版本上手动实现了一个相关任务（Santos 等人，2023年）。sivep 流水线预测 2020 年患者的严重急性呼吸道症状是否由新冠引起，并将原住民作为数据集中代表性不足的少数群体。代码训练了一个基于树的模型，并在训练集中手动应用条件过采样，以改善该代表性不足群体的预测结果。我们在该群体的测试样本上测量曲线下面积。

- 论文："Synthcity：适用于表格合成数据多种用例的基准测试框架"（Qian 等人，2023年）NeurIPS'23
- 代码仓库：https://github.com/vanderschaarlab/synthcity-benchmarking/ （由于在本地和谷歌 Colab 上反复遇到安装问题，未使用）
- 原始数据：http://plataforma.saude.gov.br/coronavirus/dados-abertos/ （现已无法获取）
- 替代数据版本：https://data.mendeley.com/datasets/f6sjz6by8k/1
- 流水线：sivep - 论文中 SIVEP 实验变体的基线重新实现
- 数据集规模： 87,000 个样本
- 指标： 针对数据中原住民少数群体预测结果的曲线下面积分数
- 实验协议：采用 50/50 训练集—测试集划分，使用五种不同的随机种子
- 实验代码： https://github.com/deem-data/sempipes/tree/v1/experiments/sivep

A.6.2. 智能体生成的流水线

我们尝试通过机器学习工程智能体 AIDE（Jiang 等人，2025a）为选定的任务生成流水线。该智能体近期在 MLEBench 中展现了最先进的性能（Chan 等人，2025年）；此外，我们还使用软件工程智能体 mini-SWE-Agent（Yang 等人，2024b）。我们向这两个智能体提供数据和自然语言指令，其中说明了任务描述和评估标准，并让它们在完全自主模式下运行。对于所有任务，我们都使用默认配置运行 AIDE，但特征工程任务除外；对于该任务，我们采用项目代码库中预先计算的 AIDE 流水线（Jiang 等人，2025b）。我们使用 mini-SWE-Agent 和 OpenAI 的 GPT-4.1 型号，每次执行的预算为 5 美元。

失败。我们概述了智能体未能生成有效流水线的任务。失败的智能体运行轨迹收录在我们的补充仓库中。

- 实体解析分块（AIDE）：
https://github.com/deem-data/sempipes/tree/v1/experiments/aide_失败_SIGMOD
- 实体解析分块（迷你 SWE-agent）：
https://github.com/deem-data/sempipes/tree/v1/experiments/swe_失败_SIGMOD
- 用于模型调试的数据标注：
https://github.com/deem-data/sempipes/tree/v1/experiments/aide_失败_公平性
- 单细胞数据整合和细胞类型预测：
https://github.com/deem-data/sempipes/tree/v1/experiments/aide_失败_单细胞数据

表6：智能体流水线生成方法产生的失败示例。

智能体	Task	失败描述
AIDE	临床信息特征提取 用于模型调试的数据标注 公平性数据增强	处理 JSON 输入文件时崩溃 因使用不存在的库而崩溃 预处理代码中的数据泄漏问题
mini-SWE-Agent	特征工程（票房） 实体解析分块 用于模型调试的数据标注	预处理代码中的数据泄漏问题 不可扩展的全对比较 返回随机属性值

A.6.3. 合成算子代码的示例结果

我们提供了合成并优化后的算子代码示例。

特征工程（Scrabble玩家评分）。

添加了语义数据算子的专家编写流水线。

```
def sempipes_pipeline():
    游戏 =skrub.var("games")

    回合 = skrub.var("回合")
    数据 = skrub.var("数据")

    def sum_first_five(系列):
        返回 求和(系列.values [::-1][:5])

    总回合数 =turns.groupby(["游戏 ID", "昵称"]).turn_number.count()
    最高分数 =回合.groupby(["游戏 ID", "昵称"]).分数.最高值()
    最低分数 = 回合.groupby(["游戏 ID", "昵称"]).分数.最小值()
    前五回合分数总和 =回合.groupby(["游戏 ID", "昵称"]).分数.agg(sum_first_five)

    游戏玩家数据 =总回合数.reset_index()
    游戏玩家数据 = 游戏玩家数据.rename(columns={"回合编号": "总回合数"})

    游戏玩家数据 = 游戏玩家数据.assign(
        前五回合得分=first_five_turn_point_sum.reset_index()["分数"],
        回合最高得分=max_points.reset_index()["分数"],
        回合最低得分=min_points.reset_index()["分数"],
    )
    游戏玩家数据 =游戏玩家数据.assign(
        max_min_difference=game_player_data.回合最高得分 - 游戏玩家数据.回合最低得分
    )

    游戏玩家数据 = 游戏玩家数据.join(游戏.set_index("游戏 ID"), 如何="左", 基于="游戏 ID")
    data = 数据.merge(游戏玩家数据, 如何="左", 左侧连接键=["游戏 ID", "昵称"], 右侧连接键=["游戏 ID", "昵称"])

    X = 数据
    y = X["评分"].skb.mark_as_y()
    X =X.drop(columns=["评分"])

    X =X.sem_gen_features(
        nl_prompt="""
        创建有助于预测玩家评分的附加特征。 这类特征可以 与.....相关 玩家如何 玩家
    在他们对局时的得分 在一回合中先手，以及 他们赢了哪些游戏， 他们赢得游戏的频率， etc.

        此外， 考虑 一名玩家通常花费多少时间 玩家通常 使用， 多少分数 每回合他们 品牌， 多少分数 每秒，
    etc.

        """,
        名称="玩家特征",
        数量=15,
    )

    编码后的 X =X.skb.应用(
        独热编码器(handle_unknown="ignore", sparse_output=False),
        列=(s.categorical() - s.cols("词典")),
    )
    编码后的 X =编码后的 X.skb.应用(
        独热编码器(类别=["NWL20", "CSW21", "ECWL", "NSWL20"], 稀疏输出=False),
        列=s.cols("词典"),
    )

    定义 drop_if_exists(df):
        要删除的列 = [列 适用于 列 在 ("创建时间", "昵称", "游戏 ID") 如果 列 在df.columns]
        返回 df.drop(columns=cols_to_drop)

    编码后的 X =编码后的 X.skb.应用函数(drop_if_exists)
    编码后的 X = 编码后的 X.skb.选择(s.numeric() |s.布尔值())
    编码后的 X = 编码后的 X.skb.应用函数(lambda x: x.replace([np.inf, -np.inf], np.nan))

    XGBoost 回归器 = XGBoost 回归器(
```

```
            按树列采样=0.4,
            伽马=0.
            学习率 =0.07,
            最大深度=3    ,
            最小子节点权重=1.5,
            估计器数量 =8000     ,
            L1 正则化系数=0.75,
            L2 正则化系数=0.45,
            子采样=0.6       ,
            随机种子=42,
            评估指标="均方根误差"
        )

        返回 编码后的 X.skb.应用(XGBoost 回归器 , y=y)
```

对应的合成算子代码。

```
# (球员场均得分)
# 实用性:          捕捉          玩家的平均值          性能          在得分方面    而这          直接          与技能/评分相关。
# 输入     样本:      '昵称':          ['BetterBot',      'stevy',      'davidavid'],     '得分':     [335, 429, 440]
df['玩家平均分'] = df.groupby('昵称')['得分'].变换('平均值')

# (玩家胜率)
# 实用性:          编码          玩家的竞技表现,                    其       应该       与较高的评分相关。
# 输入     样本:      '昵称':          ['BetterBot',      'stevy',      'davidavid'],     '获胜者':     [1, 1, 1]
df['玩家胜率'] = df.groupby('昵称')['获胜者'].变换('平均值')

# (玩家均值          分数          每回合)
# 实用性:          反映效率                    and  技能水平;       high  比率表明          出色表现。
# 输入     样本:      '得分':     [335, 429, 440], '总回合数':        [13, 14, 14]
df['player_mean_points_per_turn'] = df.groupby('昵称').apply(lambda g:(g['得分'].sum() /g['总回合数'].sum())).reindex(df.
        index).values

# (玩家                对局       游戏场次)
# 实用性:          More  游戏    比赛     may  表示经验;                    经验可以           be  与评分相关联。
# 输入样本:          '昵称':  ['BetterBot',            'stevy',      'davidavid']
df['玩家已进行的比赛数'] = df.groupby('昵称')['游戏 ID'].变换('nunique')

# (平均值游戏   时长          每位玩家)
# 实用性:          衡量          时间管理和/或对局                    打法风格,  这与    can be  评分相关。
# 输入样本:          '昵称':  ['BetterBot',            'stevy',      'davidavid'],     '游戏时长(秒)':              [674.84,    492.27,    350.86]
df['玩家平均游戏时长'] = df.groupby('昵称')['游戏时长(秒)'].变换('平均值')

# (首发游戏比例    游戏      先手)
# 实用性:          May  表明          玩家      在保障方面的偏好/技能                    先手      位置,          这      can  Impact     评分。
# 输入样本:          '昵称':  ['BetterBot',            'stevy',      'davidavid'],     '先手':  ['BetterBot',          'BetterBot',      'BetterBot']
df['玩家首发游戏占比'] =(df['昵称'] == df['先手']).groupby(df['昵称']).变换('平均值')

# (平均值   先手    five  回合    分数     per  玩家)
# 实用性:          初期          出色表现表明技能                                and  攻击性,          通常与评分有关。
# 输入     样本:      '昵称':          ['BetterBot',      'stevy',      'davidavid'],     '前五回合得分':                    [122, 95, 136]
df['玩家前五回合平均得分'] = df.groupby('昵称')['前五回合得分'].变换('平均值')

# (平均值 max     分数      在一个回合中    per  玩家)
# 实用性:          显示      玩家的峰值          性能          在单个          回合:     与评分相关。
# 输入     样本:      '昵称':          ['BetterBot',      'stevy',      'davidavid'],     '回合最高得分':          [68, 98, 103]
df['玩家每回合最高分均值'] = df.groupby('昵称')['回合最高得分'].变换('平均值')

# (平均值最小值    分数      在一个回合中    per  玩家)
# 实用性:          一致性          can be  重要          for high  评分:       更高的最低值          表示          更少的   严重失误。
# 输入     样本:      '昵称':          ['BetterBot',      'stevy',      'davidavid'],     '回合最低得分':          [8, 2, 13]
df['player_mean_min_turn_pts'] = df.groupby('昵称')['回合最低得分'].变换('平均值')

# (玩家最大最小差值均值)
# 实用性:          表示波动性                    得分方面;       波动性较低          may  关联      to  更高的技能/评分。
# 输入     样本:      '昵称':          ['BetterBot',      'stevy',      'davidavid'],     'max_min_difference':     [60, 96, 90]
df['玩家均值_max_min_diff'] = df.groupby('昵称')['max_min_difference'].变换('平均值')

# (玩家均值          分数          每秒)
# 实用性:          结合速度                    and  有效性,          提供技能                    效率代理。
# 输入     样本:      '得分':     [335, 429, 440], '游戏时长(秒)':                [674.84,    492.27,    350.86]
df['玩家平均每秒得分'] = df.groupby('昵称').应用(
    lambda g:(g['得分'].求和() /        g['游戏时长(秒)'].求和())
).reindex(df.index).values

# (平均值   初始          时间(秒)          per  玩家)
# 实用性:          捕获          首选的时间控制和                    玩家是否选择/练习更快或更慢的游戏。
# 输入     样本:      '昵称':          ['BetterBot',      'stevy',      'davidavid'],     '初始时间(秒)':              [1200, 1200,    900]
df['玩家平均初始时间'] = df.groupby('昵称')['初始时间(秒)'].变换('平均值')

# (平均值   增秒数                    每位玩家)
# 实用性:          反映          if  玩家      更喜欢     time  增量,          其      can  Impact     play  风格,     与评分相关。
# 输入     样本:      '昵称':          ['BetterBot',      'stevy',      'davidavid'],     '增量秒数':                [0, 0, 0]
df['玩家平均增时'] = df.groupby('昵称')['增量秒数'].变换('平均值')

# (每位玩家的平均最大加时分钟数     )
# 实用性:          May  表示          if  玩家习惯于                    更长      可能的      游戏,     相关于          play  风格和          评分。
# 输入     样本:      "昵称":          ["BetterBot",      "stevy",      "davidavid"],     "最大加时分钟数":              [1, 1, 5]
df['玩家平均加班时长'] = df.groupby('昵称')['最大加时分钟数'].transform('平均值')
```

```
73 74 #（玩家认输的对局比例）75 # 实用性：较高的认输率可能表明技能较低或感到挫败，并且可用于预测评分。
76
77
样本
'RES
80
81
称 ':
84 #
85
86
入样
'is_fi
将先
92
93
'昵称
'is_fi
性：
99
100
入样
'is_fi
后手
106
107
'昵称
'is_fi
112 a
 [1
laml
oc [g
在非
' 本后
'is_fi
'is_fi
与评
': '昵
tran
136
137
tran
表现
142
143
'Bett
tran
147
148
'dav
意使
152
153
'Bett
限的
157
158
本：
tran
'Bett
```

```
                                                
```

249 250 #（玩家先手时的平均加时分钟数）251 # 实用性：可能表明处于优势（先手）位置时的策略性时间使用。

252

253

时的

reinc

259

260

手时

reinc

265

266

'davi

分更

入样

272

长方

276

277

': 'Be

280 #

281

282

输入

df.so

288 2

291

292

'davidavid ' '得分 ': df '玩家得分中位数' df.groupby('昵称') '得分' .transform('median') 293 294 #（玩家每回合得分中位数）295 # 实用性：稳健地衡量玩家典型的每回合效率。

296 [ ] [25.77 30.64 31.43] # 输入样本：'昵称 ':

297

'Bett

现玩

301

302

'davi

性很

evy'、

失误

'Bett

手的

输入

胜者'

薄弱

平均

328 #

330

331

'davi

<=

335

336

'davi

性：

```
数据框['玩家_平均值_每秒得分_作为先手'] = 数据框.groupby('昵称').应用(
    lambda g:g.loc[g['本局是否先手'] == 1,['得分', '游戏时长（秒）']].assign(
        每秒得分=lambda x:x['得分'] /x['游戏时长（秒）']
    )['每秒得分'].mean() 如果存在(g['本局是否先手'] == 1) 否则 np.nan
).reindex(df.index).values

# （玩家平均 分数 每秒 非先手）
# 实用性：时间效率 没有先手优势; 稳健 玩家可能表现出 较小的下降幅度。
# 输入 样本: '得分': [335, 429, 440], '游戏时长（秒）': [674.84, 492.27, 350.86], '本局是否先手': [1, 0, 0]
df['非首位玩家平均每秒得分'] = df.groupby('昵称').apply(
    lambda g:g.loc[g['本局是否先手'] == 0,['得分', '游戏时长（秒）']].assign(
        每秒得分=lambda x:x['得分'] /x['游戏时长（秒）']
    )['每秒得分'].mean() 如果存在(g['本局是否先手'] == 0) 否则 np.nan
).reindex(df.index).values

# 清理 辅助列
df.drop(columns='每回合得分', inplace=True)
df.drop(columns='创建时间', inplace=True)
# （玩家活跃天数）
# 实用性: 度量 持续时间 and 参与度: 更长的活动 may 表明经验以及 稳定评分。
# 输入 样本: '昵称': ['BetterBot', 'stevy', 'davidavid'], '创建时间': ['2022-08-26 03:38:49', ...]
df['创建时间'] = pd.to_datetime(df['创建时间'])
df['玩家活跃天数'] = df.groupby('昵称')['创建时间'].变换(lambda x: (x.max() - x.最小值()).days + 1

# （玩家每日活跃场次平均数）
# 实用性: High game 每日频率 may 反映参与度, 专注力, 或连胜倾向, 这 可能 对评分 很重要。
# 输入 样本: '玩家已进行的比赛数': [12, 10, 8], '玩家活跃天数': [6, 5, 4]
df['玩家每日游戏局数'] = df['玩家已进行的比赛数'] / df['玩家活跃天数']

# （玩家唯一数量 对局天数）
# 实用性: 表示持续 play 而非 爆发式活动, 以便了解 体验/评分。
# 输入样本: '昵称': ['BetterBot', 'stevy', 'davidavid'], '创建时间': ['2022-08-26 03:38:49', ...]
df['玩家参与游戏的不同天数'] = df.groupby('昵称')['创建时间'].变换(lambda x:x.dt.date.nunique())

# （玩家 参与游戏的天数占比 在总数中的 活跃 天）
# 实用性: 衡量一致性 and 规律性 of 对局, 其中 can be 可用于预测 评分 提升。
# 输入 样本: '玩家参与游戏的不同天数': [5, 4, 3], '玩家活跃天数': [6, 5, 4]
df['玩家游玩天数占比'] = df['玩家参与游戏的不同天数'] / df['玩家活跃天数']

# （玩家平均 time 每回合）
# 实用性: 捕获 玩家的节奏; 较慢 play may 表示更加 深思熟虑 或缺乏经验的 对局, both 与评分相关。
# 输入 样本: '游戏时长（秒）': [674.84, 492.27, 350.86], '总回合数': [13, 14, 14]
df['每回合时间'] = df['游戏时长（秒）'] / df['总回合数']
df['玩家平均每回合时间'] = df.分组('昵称')['每回合时间'].变换('平均值')
df.删除(列='每回合时间', 原地=真)

# （玩家平均 初始 time 每回合）
# 实用性: 可能 反映 偏好 游戏节奏和时间管理。
# 输入 样本: '初始时间（秒）': [1200, 900, 1200], '总回合数': [13, 14, 14]
df['每回合初始时间'] = df['初始时间（秒）'] / df['总回合数']
df['player_mean_init_time_per_turn'] = df.groupby('昵称')['每回合初始时间'].transform('平均值')
df.drop(columns='每回合初始时间', inplace=True)

# （玩家每回合平均 增量 ）
# 实用性: 增量对对局的影响 打法风格和 结果。
# 输入 样本: '增量秒数': [0, 0, 0], '总回合数': [13, 14, 14]
数据框['每回合加时'] = 数据框['增量秒数'] / 数据框['总回合数']
数据框['玩家平均增时_per_turn'] = 数据框.groupby('昵称')['每回合加时'].变换('平均值')
数据框.drop(columns='每回合加时', 原地=真)

# （玩家最常见 词典）
# 实用性: 专业化 in a 词典可以与.....相关 技能深度 从而与评分相关。
# 输入样本: '昵称': ['BetterBot', 'stevy', 'davidavid'], '词典': ['NWL20', 'CSW21', 'NWL20']
df['玩家主要词典'] = df.groupby('昵称')['词典'].变换(lambda x: x.mode().iat[0] 如果不是 x.mode().empty 否则 x.iloc
    [0])

# （玩家最常用的时间控制）
# 实用性: 偏好 对于某些时间控制可能 形状评分。
# 输入样本: '昵称': ['BetterBot', 'stevy', 'davidavid'], '时间控制名称': ['常规', '常规', '常规']
df['玩家主要时间控制'] = df.groupby('昵称')['时间控制名称'].变换(lambda x:x.mode().iat[0] 如果不是 x.mode().
    空 否则 x.iloc[0])

# （玩家最常见的评分评级模式）
# 实用性: 表示 if 玩家主要进行额定模式 或休闲对局, 这 may 反映 认真程度/技能。
# 输入样本: '昵称': ['BetterBot', 'stevy', 'davidavid'], '评分模式': ['休闲对局', '额定模式', '休闲对局']
df['玩家主要评分模式'] = df.groupby('昵称')['评分模式'].变换(lambda x: x.mode().iat[0] if not x.mode().empty else
    x.iloc[0])

# （玩家最常见的加时赛分钟数）
# 实用性: 可能 信号 玩家偏好的 节奏和时间压力下的韧性。
# 输入 样本: '昵称': ['BetterBot', 'stevy', 'davidavid'], '最大加时分钟数': [1, 1, 5]
df['玩家主要加时'] = df.groupby('昵称')['最大加时分钟数'].transform(lambda x: x.mode().iat[0] if not x.mode().
    空 否则 x.iloc[0])

# （玩家 比赛占比 游戏 以其最常见词典进行的 词典）
# 实用性: 表示 .....的程度 专业化与多功能性之间的权衡。
# 输入样本: '词典': ['NWL20', 'CSW21', 'NWL20'], '玩家主要词典': ['NWL20', 'NWL20', 'NWL20']
df['玩家主要词典占比'] =(df['词典'] == df['玩家主要词典']).groupby(df['昵称']).变换('平均值')

# （玩家 占比 在最常用时间控制下进行的游戏）
# 实用性: 专业化 时间控制方面的专业化 may Impact 评分。
# 输入样本: '时间控制名称': ['正则表达式', '正则表达式', '正则表达式'], '玩家主要时间控制': ['常规', '常规', '常规
```

```
        ']
df['玩家占比_main_时间控制'] =(df['时间控制名称'] == df['玩家_主要_时间控制']).groupby(df['昵称']).变换(
    '平均值')

# (玩家 占比 游戏 以最常见的评级模式进行对局)
# 实用性: 一致性 比赛类型中的 可能反映认真程度。
# 输入样本: '评分模式': ['休闲对局', '额定模式', '休闲对局'], '玩家主要评分模式': ['休闲对局', '休闲对局', '休闲对局']
df['玩家占比_玩家主要评分模式'] =(df['评分模式'] == df['玩家主要评分模式']).groupby(df['昵称']).transform('平均值')

# (玩家 比赛占比 以其最常见的加时赛模式进行的已进行比赛)
# 实用性: 一致性 这里可能 与特定加班条件下的舒适度有关。
# 输入 样本: '最大加时分钟数': [1, 1, 5], '玩家主要加时': [1, 1, 1]
df['玩家主要加班比例'] =(df['最大加时分钟数'] == df['玩家主要加时']).groupby(df['昵称']).transform('平均值
    ')

df.drop(columns='创建时间', inplace=True)
# (玩家每回合平均时间 每回合为 先手)
# 实用性: 表示 if 玩家使用较多/较少 time 每回合, 且拥有先手优势, 可能 相关 with 技能/
    信心。
# 输入 样本: '游戏时长（秒）': [674.84, 492.27, 350.86], '总回合数': [13, 14, 14], '本局是否先手': [1, 0, 0]
df['每回合时间'] = df['游戏时长（秒）'] / df['总回合数']
df['作为先手时玩家每回合平均时间'] = df.groupby('昵称').apply(
    lambda g:g.loc[g['本局是否先手'] == 1, '每回合时间'].mean() 如果存在(g['本局是否先手'] == 1) 否则 np.nan
).reindex(df.index).values

# (玩家每回合平均时间 非先手)
# 实用性: 亮点 差异 in pace 取决于 on play 顺序, 相关的 用于风格/评分。
# 输入 样本: '游戏时长（秒）': [674.84, 492.27, 350.86], '总回合数': [13, 14, 14], '本局是否先手': [1, 0, 0]
df['玩家非首回合平均用时'] = df.groupby('昵称').应用(
    lambda g.loc[g['本局是否先手'] == 0, '每回合时间'].平均值() 如果存在(g['本局是否先手'] == 0) 否则 np.nan
).reindex(df.index).values

# (先手与非先手的每回合平均时间比 每回合 先手与非先手)
# 实用性: 直接 比较 时间使用情况; 高/低比率可能 区分 不在先手时的信心或不适。
# 输入样本: '作为先手时玩家每回合平均时间': [40.3, np.nan, np.nan], '玩家非首回合平均用时': [np.nan, 43.2,
    41.7]
df['玩家先手与非先手每回合时间'] =(
    df['作为先手时玩家每回合平均时间']/ df['玩家非首回合平均用时']
)

# (玩家均值 初始 time 每回合先手时 先手)
# 实用性: 显示是否 初始时间分配 每回合 is 不同的情况 先行 先手。
# 输入 样本: '初始时间（秒）': [1200, 900, 1200], '总回合数': [13, 14, 14], '本局是否先手': [1, 0, 0]
df['每回合初始时间'] = df['初始时间（秒）']/ df['总回合数']
df['作为先手时玩家每回合平均初始化时间'] = df.groupby('昵称').apply(
    lambda g:g.loc[g['本局是否先手'] == 1, '每回合初始时间'].mean() 如果任意(g['本局是否先手'] == 1) 否则 np.nan
).reindex(df.index).values

# (玩家均值 初始 time 每回合不 先手)
# 实用性: 差异化 策略 按时间分配 基于 on play 顺序。
# 输入 样本: '初始时间（秒）': [1200, 900, 1200], '总回合数': [13, 14, 14], '本局是否先手': [1, 0, 0]
df['玩家非首回合平均初始时间'] = df.groupby('昵称').应用(
    lambda g:g.loc[g['本局是否先手'] == 0, '每回合初始时间'].平均值() 如果存在(g['本局是否先手'] == 0) 否则 np.nan
).reindex(df.index).values

# (比率 mean 初始 time 每回合 先手到非先手 先手)
# 实用性: 比较 初始时间分配 模式, 可能是 与评分相关 信心。
# 输入样本: '玩家每回合平均初始化时间_as_first': [100, np.nan, np.nan], '玩家非首回合平均初始时间': [np.
    nan, 110, 107]
df['先手与非先手玩家每回合初始化时间'] = (
    df['作为先手时玩家每回合平均初始化时间']/ df['玩家非首回合平均初始时间']
)

# (玩家均值 增量 每回合的 先手)
# 实用性: 记录 如果增量策略 因对局 顺序。
# 输入 样本: '增量秒数': [0, 0, 0], '总回合数': [13, 14, 14], '本局是否先手': [1, 0, 0]
df['每回合加时'] = df['增量秒数']/ df['总回合数']
df['作为先手时的玩家每回合平均增量'] = df.groupby('昵称').apply(
    lambda g:g.loc[g['本局是否先手'] == 1, '每回合加时'].mean() 如果存在任意(g['本局是否先手'] == 1) 否则 np.nan
).reindex(df.index).values

# (玩家均值 增量 每回合非先手)
# 实用性: 按行动顺序分段, 提取更细致的特征 模式。
# 输入 样本: '增量秒数': [0, 0, 0], '总回合数': [13, 14, 14], '本局是否先手': [1, 0, 0]
数据框['玩家非首回合平均每回合增量'] = 数据框.groupby('昵称').应用(
    lambda g:g.loc[g['本局是否先手'] == 0, '每回合加时'].平均值() 如果有(g['本局是否先手'] == 0) 否则 np.nan
).reindex(df.index).values

# (比率的 mean 增量 每回合作为 先手与非先手)
# 实用性: 差异增量 利用 模式 按对局 行动顺序可以 be 关联 with 评分。
# 输入样本: '作为先手时的玩家每回合平均增量': [0.0, np.nan, np.nan], '玩家非先手平均每回合增量': [np.
    nan, 0.0, 0.0]
df['玩家先手与非先手时的每回合增量'] = (
    df['作为先手时的玩家每回合平均增量']/ df['玩家非首回合平均每回合增量']
)

df.drop(columns=['每回合时间', '每回合初始时间', '每回合加时'], inplace=True)
# (玩家 获胜比例 by 认输)
# 实用性: 获胜 by 认输可能 表明 优势或 威吓 因素, 这可能 can 相关 with 评分。
# 输入 样本: '昵称': ['BetterBot', 'stevy', 'davidavid'], '获胜者': [1, 1, 1], '对局结束原因': ['标准', '认输', '
    标准']
df['玩家认输获胜比例'] = df.groupby('昵称').apply(
```

```
513 lambda g: ((g['获胜者'] == 1) & (g['对局结束原因'] == '认输')).sum() / max((g['获胜者'] == 1).sum(), 1) 514 ).reindex(df.index).values 515 516 #（玩家因认输而输掉的比例）
517 #
因':
max
分。
时长
527

g.loc
败可
533
534
（秒
] ==
540
541
分' d

lamb
性：
548
549
对局
rein
555
556
入样
& (g
输失
563
564
入样
& (g
df.dr
因素
572
573
称':
575 #
                                [1, 1, 0],                                [                                ]# 输入样本：'获胜者'：'对局结束原因'：'标准'、'认输'、
[                                ] =
'标准
（玩
584
585
'标准
588
                                [1, 0, 0],                                [                                ]# 输入样本：'获胜者'：'对局结束原因'：'认输'、'认输'、
'标准
596 #
598
599
样本
```

```
600 df['玩家平均回合最高得分占比'] = df.groupby('nickname')['回合最高得分占比'].transform('mean') 601 df.drop(columns='回合最高得分占比', inplace=True) 602 603 #（玩
家得
605
606
df '得

trans
能是

分':

比率'
平均
619
620
' df '
大值
626
627
df['玩
差）
633
634
pd.to

分' .t
可以

'游戏
df.gr
真实

入样
not x
654 #
'Bett
mea
'Bett

664 l
能会
670

与对
676
677
'stev
rein
Bette


684
df.gr
```

```
).reindex(df.index).values

# （玩家比赛场次    已进行的比赛数与不同对手数）
# 实用性:        High  数字表示深度            的经验。
# 输入样本:          '昵称': ['BetterBot',          'stevy',    'davidavid'],   '对手': ['stevy',          'BetterBot',    'BetterBot']
df['玩家对阵不同对手的比赛数'] = df.groupby('昵称').应用(
    lambda g:g.groupby('对手').size().mean()
).reindex(df.index).values

# （玩家平均得分作为          先手 vs 主要对手)
# 实用性:          性能作为          先手 vs 对手      may  信号     信心和          技能。
# 输入   样本:      '本局是否先手': [1, 0, 1], '对手':      ['stevy',    'BetterBot',    'BetterBot'],    '主要对手玩家':          ['stevy',
      'BetterBot'  , 'BetterBot']  , '得分': [335  , 429, 440]
数据框['玩家先手时对阵主要对手的平均得分'] = 数据框.groupby('昵称').应用(
    匿名函数 g: g.loc[(g['本局是否先手'] == 1) &(g['对手'] ==                  g['玩家的主要对手'].iloc[0]),          '得分'].mean()
    如果存在任意((g['本局是否先手'] == 1) &(g['对手'] ==            g['主要对手玩家'].iloc[0]))否则 np.nan
).reindex(df.index).values

# （玩家非先手对阵主要对手时的平均得分）
# 实用性:          性能          来自第二个     与主要对手相比的行动顺序     对手。
# 输入   样本:      '本局是否先手': [1, 0, 1], '对手':      ['stevy',    'BetterBot',    'BetterBot'],    '主要对手玩家':          ['stevy',
      'BetterBot'  , 'BetterBot']  , '得分': [335  , 429, 440]
数据框['player_mean_score_not_first_vs_main_opponent'] = 数据框.groupby('昵称').应用(
    lambda g: g.loc[(g['本局是否先手'] == 0) &(g['对手'] ==                  g['主要对手玩家'].iloc[0]),          '得分'].平均值()
    如果存在((g['本局是否先手'] == 0) &(g['对手'] ==            g['主要对手玩家'].iloc[0]))否则 np.nan
).reindex(df.index).values

# （玩家先手时对阵主要对手的胜率    先手 vs 主要对手）
# 实用性:          展示    玩家的能力          在.....时获胜    先手对阵其最常遇到的                对手。
# 输入   样本:      '本局是否先手': [1, 0, 1], '对手':      ['stevy',    'BetterBot',    'BetterBot'],    '主要对手玩家':          ['stevy',
      'BetterBot'  , 'BetterBot']  , '获胜者': [1  , 1, 0]
df['玩家作为先手对主要对手的胜率'] = df.groupby('昵称').apply(
    lambda g.loc[(g['本局是否先手'] == 1) &(g['对手'] ==                  g['主要对手玩家'].iloc[0]),          '获胜者'].mean()
    if any((g['本局是否先手'] == 1) &(g['对手'] ==            g['主要对手玩家'].iloc[0]))else np.nan
).reindex(df.index).values

# （玩家非先手时对主要对手的胜率）
# 实用性: 同上,          但不包含先手优势。
# 输入   样本:      'is_first_this_game': [1, 0, 1], '对手':      ['stevy',    'BetterBot',    'BetterBot'],    '主要对手玩家':          ['stevy',
      'BetterBot'  , 'BetterBot']  , '获胜者': [1  , 1, 0]
df['非先手对阵主要对手时的玩家胜率'] = df.groupby('昵称').apply(
    lambda g: g.loc[(g['本局是否先手'] == 0) &(g['对手'] ==                  g['主要对手玩家'].iloc[0]),          '获胜者'].mean()
    如果存在((g['本局是否先手'] == 0) &(g['对手'] ==            g['主要对手玩家'].iloc[0]))否则 np.nan
).reindex(df.index).values

df.drop(columns=['对手',          '主要对手玩家'],          inplace=True)
```

SIGMOD 2022 编程竞赛 - 阻塞。

e 的特征提取部分   专家编写的 *rutgers* 流水线，已添加语义数据算子。我们仅展示特征提取   *n* 由于篇幅限制。

```
def 提取特征(数据: pd.DataFrame)          -> pd.DataFrame:
    """
    提取特征          使用 sempipes 和 sem_extract_features 从产品数据中提取。
    返回一个包含          合并后的提取特征          with 原始数据的 DataFrame。
    """
    流水线 =skrub.var("data",          数据).skb.mark_as_X()
    流水线 =    流水线.skb.set_description(
        "这   is a 数据集的      产品标题的          可移动存储          设备      "
        "(USB   存储棒,  SD / MicroSD   卡,  固态硬盘,  存储器   卡,  and  有时      三星手机          / 电视）。"
    )

    品牌列表 = ['Intenso',      'lexar', 'LogiLink',   'pny', '三星',      '闪迪',      '金士顿',      '索尼',  '东芝',      '创见']
    families_dict ={
        '闪迪':  ['Cruzer',      '卡',      '滑动',      '选择',      '外部',      '起源',      '跨记忆',      'React',  '备忘录',  '卡丁车',
                  'U盘',      '汽车',  '系列',  '线',    '极限',      'Cruzer',  'Ultra',  '微型',  '旅行者',      'HyperX',
                  '适配'  , 'WEX'  , '闪存']  ,
        'Lexar':  ['Ultra',  'XQD',  '闪存盘',      '微型',  'U盘',      'SD卡',  '卡',      '备忘录',  'USB',  '极限',      '刀片',
                  '汽车'  , '卡'      , 'veloc'  , 'React'  , '适配'  , 'secure'  , '高端'      , 'WEX'  , 'transmemo'  , 'alu'  ,
                  'datatravel', '画布',      '风格',    'HyperX',  'Cruzer',  '闪存'],
        '东芝':  ['Ultra'      , 'Exceria'  , '旅行者'      , 'SDHC'  , '记忆'      , 'XQD'  , '线'    , 'USB'  , 'transmemo'  , '极限'      ,
                  '风格'  , '微型'  , '存储器'  , '系列'  , '汽车'] ,
        '金士顿':  ['旅行者',      '购物车',  '适配',    '极限',      '备忘录',  '画布',      'datatravel', 'HyperX',  '卡丁车',  '刀片',
                  '终极']  ,
        '索尼':  ['极限',      'USM32GQX',  '微型',    'SD卡',  'USB',  'Ultra',  '闪存盘',      'HyperX',  '备忘录',  '卡丁车',  'XQD',
                  'U盘',      '适配',    '刀片',    'Cruzer',  '风格',    '滑动',    '购物车',  '卡',      '闪存'],
        'Intenso':  ['cs/超高',      '高端',      'Ultra',  '微型',    '线',    '卡',      'USB',  'SD卡',  '卡',      '卡丁车',  '汽车',
                  'transmemo'] ,
        'PNY':  ['附件',    'USB',  'SD卡',  '微型',    '高端',      '备忘录'],
        '三星':  ['Galaxy',      '存储器',      '微型',    'USB',  'SD卡',  'evo', 'Ultra',  '极限',      '备忘录',  '适配',    '汽车',  '卡丁车',
                  '等级',    '多功能',  '闪存盘',      '闪存'],
        '创见':  [],
    }

    Intenso 类型 = ['基础型',    'Rainbow',  "高    速度",  "速度",  "高端",      'alu', "商务",      '微型',    'imobile',
                  'cmobile'  , '迷你'  , 'Ultra'  , '纤薄'  , '闪存'  , '移动']

```

```
    颜色  = ['午夜黑', '棱镜白', '棱镜黑', '棱镜绿', '棱镜蓝', '金丝雀黄', '火烈鸟粉', '主教红', '烟蓝', '深蓝', '珊瑚橙', '黑天', '金砂', '蓝雾与桃云', '兰花灰', '金属铜', '薰衣草紫', '海洋蓝', '
```

```
            "宽松处理——如果看到 'extreme pro'，提取 'extreme '。如果看到 'cruzer fit'，提取 'cruzer '。" "只有在非常确定没有类型/线别信息时，才返回 '0'。" ), "model": ( "提
```

```
            “这 对于处理 处理 多语言 文本。”
            "\n\n语言规范化要求：\n"
            "1. 容量单位：归一化 'go', 'go', '吉字节' ->'GB'; 'MO', 'MO' -> 'MB'."
            " 处理所有语言变体的 容量 单位（法语 'go', 西班牙语 'GB', 等等）。"
            "2. 产品类型同义词：规范化 '存储卡'/'卡'/'卡 '/'卡'/'存储卡' -> '卡 '; "
            " 'USB密钥'/'密钥'/'U盘 '/'USB存储器' -> 'USB' or '存储棒 '."
            "3. 速度/等级: 标准化 '10类'/'10类'/'10 类'/'c10' -> '10级 '; "
            " 'UHS-I'/'UHS-I'/'UHS I' -> 'UHS-I '."
            "4. 所有文本应为 归一化后的小写。"
            "\n\n提取需求：\n"
            "1. 遵循正则表达式 模式和 清洗 逻辑 在每个.....中提供 输出列 描述 完全一致。"
            "2. 这些模式是 来自经过验证的 阻塞代码 - 请原封不动地使用它们, 不要修改或简化 它们。"
            "3. 对于每个字段, 跟随 the 精确 提取 逻辑 描述 （findall 与搜索, 去重方法, 排序,
        连接）。”
            "4. 应用 别名替换 完全按照规范化名称中的指定 描述。"
            "5. 预先编译所有 正则表达式 使用 re.compile() 编译模式 以提升性能。"
            "6. 使用指定的 方法：当 描述中写着 “find all”, 当使用 re.search() 时 它写道 '先手匹配'。"
            "7. For 去重：使用集合 然后转换回排序列表。"
            "8. 对于连接：使用单个空格 ' '。join()。"
            "9. 返回 '0' （字符串零） 用于缺失值, NOT 空 字符串或 None。"
            "\n\n对召回率至关重要 - 最高优先级：\n"
            "Aim 为了达到最高召回率 - 提取一些内容（即使稍有 瑕疵）也比返回 「0」更好。"
            "For 阻塞实体 匹配, 缺少一个 特征（返回 “0”）会导致遗漏 匹配项，并大幅 降低召回率。"
            “只要某个模式哪怕稍微有点匹配, 就提取出来。 Only 只有在 '0' 你绝对确定不存在
        匹配时才返回。”
            “处理多语言 变化 极限地 激进地 - 提取时 in 怀疑, 提取 部分 匹配项, 提取
        变体。”
            “优先提取 多个候选项，而不是返回 '0'. 如果不确定 在两个值之间, 请提取两者（以空格分隔）。
        ”
            "For 阻塞, false 正面结果可以接受 但负面结果（遗漏 特征）会严重影响 召回率。"
            "\n\n型号提取：\n"
            "积极提取型号 编号/库存单位编号 使用模式：带连字符或斜杠的字母数字代码 ”
            "（例如, 'SDCZ50 -064G-B35', 'MB-MG32DA/EU', 'SDSQUNC -032G-GN6IA'）。"
            "这些通常是最强的匹配信号。"
            “规范化 by 移除多余空格并 确保 一致的格式。”
            “查找 模式 例如: [A-Z]{2,}[0-9A-Z-]+ 或类似的字母数字代码。”
            "\n\n文本清理：\n"
            "移除 网址, HTML 标签, 卖家 信息（例如, 'amazon.es:', 'fnac.es', 'tesco Direct:') 更积极地。“
            “清理空白字符并 特殊 字符。”
            "\n\n性能: 优化 以提高吞吐量 - 必须处理数百万个 15 分钟内的行。”
            “预编译正则表达式 模式, 避免 重复 编译, 使用高效的 字符串 操作。"
            "\n\n在注释中, 解释 你的代码并 你选择这种方法的原因 你做到了。”
    ),
    输入列=["名称"] ,
    名称="提取特征" ,
    输出列=输出列 ,
    通过代码生成=是,
)

流水线 = 流水线.sem_gen_features(
    nl_prompt=(
        "您正在分析 多语言 产品标题的 存储 设备（U盘, SD卡, 固态硬盘, 等） 用于实体 阻塞
    /匹配。"
        "The 目标是 找出重复项 可能的产品 be 以不同方式描述 in 不同的语言或采用不同的
    格式。”
        "\n\n已提取的现有列: 品牌, 容量, 规范化名称, 内存类型, 类型, 型号, 长型号,
    短型号 , "
        "特征, 物品代码, 系列, pat_hb, 混合型, 长数字。"
        "这些用于 屏蔽模式 例如: 品牌+容量, 品牌+容量+类型, 品牌+型号, 品牌+特征, ”
        "长型号, 短型号, 混合型, 长数字, 系列, 物品代码。"
        "\n\n任务：生成 3-5 个有助于匹配重复商品的附加特征 跨产品 不同语言中。"
        "专注于 那些 常见的特征 当前（占>10%的 行）。 语言无关, and can 可通过.....提取 使用 正则表达式/规则
    -基方法。"
        "\n\n高优先级候选项：\n"
        "1. 速度/等级标识: 提取并 规范化 '10 类'/'10类'/'10类'/'C10' to '10级 '; ”
        " 'UHS-I'/'UHS-I'/'UHS-I'/'UHS I' to 'UHS-I '; 速度，单位为 MB/s（标准化单位: 'MB/s', '兆比特每秒', '兆字节每秒' to '兆比特每秒 ')."
        "2. 外形规格：规范化 'MicroSD'/'micro SD'/'micro -sd' to 'MicroSD '; 'SDXC '/'SD XC' to 'SDXC '."
        " 提取 保持一致: 'MicroSD ', 'microSDHC ', 'microSDXC ', 'SD卡', 'SDHC', 'SDXC', 'USB', 'XQD', 等。"
        "3. 包装/套装 信息: 提取 '套装', '包', '一包', '一批', 等等 "
        "4. 适配器信息: '带适配器'/'带适配器 '/'带适配器' 转换为布尔值或规范化文本。"
        "5. 颜色 信息: 提取 颜色名称 可能有助于 区分 产品。"
        "\n\n生成 用于提取这些内容的 Python 代码 特征，使用 正则表达式和 基于规则的 方法。"
        "返回 特征作为 小写字符串, '0' 用于处理缺失值。"
        "优化 以提高吞吐量 - 必须处理数百万个 行高效地。”
    ),
    名称="发现其他阻塞特征",
    how_many=5,
)

result_df = pipeline.skb.评估()

返回result_df
```

用于特征提取的对应合成算子代码。

```
导入 pandas 作为pd
导入 正则表达式
导入 Unicode 字符数据
来自 tqdm.auto 导入 tqdm # 使用 tqdm.auto 以兼容笔记本，或者直接使用 'tqdm'
```

```
# --- 重要：必须严格遵守 ---
# 明确规定：“禁止使用 Transformers 库。只能使用正则表达式和基于规则的方法，
# 请勿使用任何 Transformer，NER，大语言模型回退，或语言模型。”
# 所有参考资料 to `Transformers`，`PyTorch`，`流水线`，`选择设备`，and the `设备` 变量
# 来自初始样板代码是已移除，以严格遵守此禁令。

# 预编译所有正则表达式模式并定义模块级辅助数据性能级别。

# --- 0. 规范化别名和模式 ---
# This list holds（模式_to_match_正则表达式_string，替换字符串）元组。
# 更长、更具体的内容至关重要模式会优先于较短的通用模式进行处理
# 使用顺序的`re.sub`以实现正确的文本转换。
原始规范化别名模式 = [
    (r'Ultra 高速', 'UHS-I'),  # 更长，更具体用于 UHS-I
    (r'uhs1', 'UHS-I'),
    (r'uhs-i', 'UHS-I'),

    (r'类型c', 'Type-C'),  # 各种 Type-C USB 的形式
    (r'类型 c', 'Type-C'),
    (r'USB-C', 'Type-C'),
    (r'USB-C', 'Type-C'),

    (r'HyperX', 'kingston hxs'), # 'HyperX Savage' -> 'Kingston HXS' 适用于特定产品线
    (r'Savage', '金士顿 hxs'),

    (r'UX 系列', '索尼 g1ux'),  # 特定 Sony 系列别名

    (r'dtig4', ' 金士顿 DT101 '), # 别名 for 金士顿 DataTraveler 101/G2
    (r'\s101\s', ' 金士顿 DT101 '),
    (r'dt101g2', ' 金士顿 DT101 '),

    (r'sda10', ' Kingston Ultimate '), # 别名 for 金士顿 Ultimate 系列
    (r'sda3', ' Kingston Ultimate '),

    (r'极限\s', '极限 '), # 标准化 '极限' 拼写
    (r'附件', 'att4'), # '附件' -> 'att4' 按照 PNY Attache 系列的提示

    # 语言规范化要求：容量单位，产品类型同义词，速度/等级
    (r'存储卡', '卡'),  # 法语 「存储卡」
    (r'钥匙 USB', 'usb 存储棒') , # 法语 「USB 闪存盘」（U盘）
    (r'键', 'usb 存储棒'),  # 更短版本，已处理在多词之后
    (r'U盘', 'usb 存储棒'),  # 通用术语
    (r'USB存储器', 'USB 闪存盘'),  # 西班牙语 'USB存储器'

    (r'\b卡\b', '卡'),  # 独立 '卡' 各种语言
    (r'\b卡\b', '卡'),  # 西班牙语 '卡'
    (r'\b卡\b', '卡'),  # 波兰语/其他 '卡'
    (r'\b存储卡\b', '卡'),  # 瑞典语 '存储卡 卡'

    (r'类别\s*(10)', r'类别\1'),  # 针对具体数字。统一 '10类' -> '10 类'
    (r'10类', r'类别\1'),  # '10类' -> '10 类'
    (r'cl\s*10', r'10级'),  # '10级' -> '10级'
    (r'类别(e |a)?\s*(?=\d{1,2})', r'类别'),  # 通用模式 '类别', '类别' 等等到 '类别' 之前数字。

    (r'吉字节', 'GB'),  # 法国 '吉字节'
    (r'\bgo\b', 'GB'),  # 法语 千兆字节单位

    (r'\bMO\b', 'MB'),  # 法国 兆字节单位

    (r'太字节', 'TB'),  # 法国 '太字节'
    (r'\bto\b', 'TB'),  # 法语太字节单位
]

# 排序模式按字符串按长度降序应用较长／更具体的替换先手。
原始规范化别名模式.sort(key=lambda x:len(x[0]), reverse=True)
# 编译全部模式以提高效率期间字符串替换操作。
已编译的规范化别名 = [(re.编译(pat, 标记=re.IGNORECASE), 替换)
                      用于 pat，repl 在 NORMALIZATION_ALIASES_RAW_PATTERNS ]

# 5. 移除正则表达式 - 已复制 逐字来自提示。此模式针对各种 HTML 实体，
# 分隔符，特殊字符，and 某些特定上下文清理。
RE_NORMALIZED_NAME_REMOVAL =正则表达式。编译(
    正则表达式'&(nbsp|与符号|注册商标符号|[a-z]?尖音符|引号|商标符号);?|[];:+©\\(\\) \\]\\[psn ]* |(?<=USB)[\\s][m ]*(?=[23][\\.\\s]) |(?<=USB)-[\\w]+\\s
      (?=[23][\\.\\s]) |(?<=[a-z])[\\s]+GB |(?<=数据 |跳跃)[t\\s](?=旅行 |驱动器) |(?<=极限 |Exceria)[\\s](?=专业版[\\s]|增强版) |(?<=类别)
      [\\s_](?=10|[234]\x08) |(?<=世代)[\\s_](?=[134\\s][0]*)',
    标记=re.IGNORECASE
)

# 6. 10级替换正则表达式 - 已复制 逐字原样。这专门针对 "10 类" 变体。
RE_CLASS10_REPLACEMENT =正则表达式。编译(
    r'(10 类别|类别 10|类别(?=[\w]+10\x08)|cl\s10)',
    标记=re.IGNORECASE
)


# --- 品牌提取 ---
# 模糊匹配 应规范为标准品牌的别名名称在主提取正则表达式之前。
# 这 此列表用于积极品牌召回率至处理拼写错误和空格问题。
品牌模糊替换 = [
    (正则表达式。编译(r'\bsan\s?disk\b|\bsan-disk\b|\bsandisc\b|\bsan-disc\b'), '闪迪'),
    (正则表达式.compile(r'\bsamsun\b'), <style id="1">'三星'),
    (正则表达式.compile(r'\bkingstn\b'), '金士顿'),
```

96 (re.compile(r'\btoshbia\b'), 'toshiba'), 97 (re.compile(r'\btranscent\b'), 'transcend'), 98 (re.compile(r'\bhynix\b'), 'hynix'), # 常见品牌 99 (re.compile(r'\balexa\b'), 'lexar') #
Lexa
logil
使单
一组
116
121
'mer
124
'jum
'can
'mic
'blac
127
'sony
128
'tarj
129
'atta
usb',
为空
'mob
blue
copper', 'lavender purple', 'ocean blue', 'pure white', 'alpine white', 'copper', 'red', 'black', 'blue', ' white', 'silver', 'gold', 'violet', 'purple', 'brown', 'orange', 'coral', 'pink']135 136 #
汇总品牌专属 Intense 和三星列表中的所有潜在类型 / 系列术语 137 # 还包含提示中的原始模式，以拉大匹配范围 138 TYPE_SERIES_RAW_PATTERNS = [139 r'datatraveler'
r'extreme [p]?', r'exceria [p]?', r'dual [\s]*(?!=sim)', r'evo', r'xqd', r'ssd', r'cruzer [\w+]*', 140 r'glide', r'blade', r'basic', r'fit', r'force', r'basic line', r'jump\s?drive', r'hxs',
r'rainbow', r'speed line', 141 r'premium line', r'att4', r'attach', r'serie u', r'r-serie', r'beast', r'fury', r'impact', r'a400', r'sd [hx]c', 142 r'uhs [i12] [i]*', r'note\s?9', r'ultra',
r'pre
先匹
正则
TYPE
行长
特定
lsd [
', 161
re.co
re.co
176

```
RE_FEATURES =re.编译(
    r'\b(USB[23]|类型 -c|UHS[i]{1,2}|类别[0134]{1,2}|世代[1-9]{1,2}|u[23](?=[\s\.])|SD卡|hx|c|OTG|LTE|[45]g[-\s]LTE|[0-9]+(?=-inch)
    |[0-9]{2,3}r[0-9]{2,3}w|[0-9]{2,3}(?=[\smbo/p]{3}))\b',
    flags=re.IGNORECASE
)


# --- 商品编码提取 ---
# 正则表达式用于 find explicit 个数字 ic 商品代码, 通常 包含在 括号中, 并且 可能带有前缀 with 'mk'。
RE_ITEM_CODE =正则表达式.编译(r'(\((mk)?[0-9]{6,10}\))', 标记=re.IGNORECASE)


# --- 连字符模式 (PAT_HB) 提取 --- # R
用于提取任意内容的正则表达式 hyph伊纳特d alph阿努梅尔ic 模式。
RE_PAT_HB =正则表达式。编译(r'\w+-\w+', 标志=re.IGNORECASE)


# --- 混合提取 ---
# 用于提取的正则表达式 '混合型' 标识符：字母数字 长度为 5+的模式，其中包含
# 至少一个 字母，且至少一个 数字。 这是一种常见且 稳健的 模式 型号 代码。
RE_HYBRID = 正则表达式.编译(r'\b(?=.*\d)(?=.*[a-zA-Z])[a-zA-Z0-9]{5,}\b', 标记=re.IGNORECASE)


# --- 长数字提取 ---
# 用于查找任意连续 4 个或更多数字的正则表达式。
RE_LONG_NUM =正则表达式.编译(r'[0-9]{4,}')


# --- 通用辅助函数 ---
def _process_matches(matches, 默认值='0'):
    """
    去除 a 的重复项 列表进行去重 匹配的词元, 转换 将它们转换为 小写, 排序 them 按字典序,
    并用单个空格将它们连接起来。 返回指定的 `默认值` ('0') 如果没有有效的 匹配项 找到。
    """
    如果没有 匹配项:
        返回 默认值
    # 转换所有 匹配项为字符串, 筛选掉任何 None/空 值, 然后转换为 小写。
    # 集合 移除 重复项, and 排序()、 确保一致的 输出 顺序。
    唯一匹配项 =排序(列表(集合(字符串(m).lower() 对于 m在 匹配项 如果 m)))
    返回 ''.join(唯一匹配项) 如果 唯一匹配项 否则 默认值


# --- 特征提取函数 ---

定义 文本规范化(名称: 字符串) ->字符串:"""

    应用多步骤清洗和 归一化 处理流程于 输入 产品名称。
    This 包括 ASCII 转换, 小写化, 应用 各种别名和 替换,
    and 通用文本清理, 确保 一致性 以便后续进行特征提取。
    """
    if not isinstance(名称, 字符串):
        名称 =字符串(名称)

    # 1. & 2. 将输入转换为 ASCII 字符并 移除 变音符号（重音符号）。
    规范化名称 =Unicode 数据规范化('NFKD', 名称).encode('ASCII', '忽略').decode('utf-8')

    # 3. 将文本转换为小写 以进行不区分大小写的匹配。
    规范化名称 = normalized_name.lower()

    # 4. 使用编译后的预定义别名替换列表 正则表达式模式。
    # 模式为 按其长度顺序处理 长度 （最长的优先），以避免 部分 匹配项。
    用于模式正则表达式, 替换字符串在已编译的规范化别名:
        规范化名称 = 模式正则表达式.sub(替换字符串, 规范化名称)

    # 5. 应用 广泛的 移除 正则表达式以消除各种特殊 字符, HTML 实体, and 噪声。
    规范化名称 = RE_NORMALIZED_NAME_REMOVAL.sub('', 规范化名称)

    # 6. 应用特定的正则表达式进行规范化 '10 类' 变体到 '10级'。
    规范化名称 = RE_CLASS10_REPLACEMENT.sub('10级', 规范化名称)

    # 7. 应用 特定字符串 替换 for 进一步的一致性 （例如, 合并 "class X" to "classX"）。
    规范化名称 = normalized_name.replace('类别 4', 'class4')
    规范化名称 = 规范化名称.replace('3类', 'class3')
    # 同时标准化常见的产品术语 以便后续提取更简洁。
    规范化名称 = 规范化名称.replace('微型 SD卡', 'MicroSD')
    规范化名称 = 规范化名称.replace('微型 SD 卡', 'MicroSD 卡')
    规范化名称 = 规范化名称.replace('SD卡', 'SD 卡')


    # 8. 规范化所有 空白字符 by 将多个 空格缩减为单个空格, 并 去除 首尾空白。
    当 ' ' 位于 规范化名称:
        规范化名称 = 规范化名称.replace('  ', ' ')

    # 返回 '0' 如果清理后的 字符串 变为空, 表示没有有意义的 内容。
    返回 规范化名称.strip()如果 规范化名称.strip() 否则 '0'


def _extract_brand(规范化名称: 字符串) -> 字符串:
    """
    提取 主要产品品牌。 它首先应用一组模糊 替换来规范
    通用 拼写错误或空格问题 问题 in 品牌 名称（例如, "san 磁盘" to "闪迪"）。 然后应用一个
    针对全面的严格正则表达式 已知品牌 列表。 这种方法 可最大限度提高召回率。
```

```
        """
        如果 规范化名称 == '0':
            返回 '0'

        # 应用 模糊匹配 品牌 在临时字符串上进行替换 字符串 以避免 修改 原始 `规范化名称`。
        用于品牌提取的临时名称 =规范化名称
        用于模式正则表达式, 替换字符串在品牌模糊替换:
            用于品牌提取的临时名称 = 模式正则表达式.sub(替换字符串, 用于品牌提取的临时名称)

        # 查找所有 出现于 已识别的品牌 名称 来自（可能经过 模糊匹配更正的）文本。
        匹配项 = 品牌正则表达式.findall(用于品牌提取的临时名称)
        返回 _process_matches(匹配项)


    def 提取容量(规范化名称: 字符串) -> 字符串:
        """
        提取存储容量 来自名称。 它结合使用多种策略:
        1. 显式识别数字 后跟 单位术语（例如, '32GB', '128GO', '1TB').
        2. 对于常见的独立数字（例如, '64', '256'), 它会积极地 假设 'gb' 如果没有
           已被捕获的显式单位 由 策略 策略1针对该数字。 This 提高召回率。
        单位会进行规范化（例如, 'go' to 'GB', 'to' to 'TB'）。
        """
        如果 规范化名称 == '0':
            返回 '0'

        容量 =设置()

        # 策略 1: Find 明确的数字 后跟 已识别 单位术语。
        # 正则表达式 RE_CAPACITY_COMBINED 现在正确地 包含第二个捕获组 用于单位。
        用于 匹配在 RE_CAPACITY_COMBINED.finditer(normalized_name):
            数量 = match.group(1) # 例如, '32'
            单位 = match.group(2).lower() # 例如, 'GB', '前往', 'g'

            if 单位 在 ['GB', '继续', 'g', '零工']:
                容量.add(f"{num}GB")
            否则 单位 在 ['TB', 'to', 't', '太拉']:
                容量.add(f"{num}TB")

        # 策略 2: 识别 独立的常见容量单位 数字并推断 'gb' 如果未找到显式单位。
        词元 = 规范化名称.split()
        对于i, 词元 在 enumerate(tokens):中
            如果 词元.isdigit()并且 词元 在 COMMON_CAPACITY_NUMBERS:
                # 检查此数值 部分是否已经 被捕获 as 作为单位的一部分（例如, '16GB')。
                # 我们构造一个合成表示, 描述该单位若已 提取出来时的样子。
                # 示例: If 容量 has '16gb' and `词元` is '16', then this `已被明确捕获` is 是。
                已被明确捕获 =否
                针对 现有容量 在 容量中:
                    如果 existing_capacity.startswith(词元)并且 (existing_capacity.endswith('GB')或 existing_capacity.endswith('TB')):
                        is_already_explicitly_captured =真
                        跳出

                如果不是 is_already_explicitly_captured:
                    # 如果该数字是常见容量单位, and 并非立即 后跟 特定的其他单位（例如 MB),
                    # 积极地 假设 'gb' to 提升 针对这些情况的召回率 like "闪迪 Ultra 128"。
                    # 避免假设 'GB'（如果明确） 后面跟着 'mb' or 'MO'。
                    如果 i + 1 < 长度(词元):
                        如果任意(u 在 词元[i+1].lower() 对于u 在 ['MB', 'MO']):
                            继续
                    capacities.add(f"{token}GB")# 积极添加为 'gb'

        返回 _process_matches(列表(容量))


    def _extract_mem_type(规范化名称: 字符串) -> 字符串:
        """
        提取 不同的内存产品类型 基于 on a 层次逻辑。 It 优先处理 直接
        正则表达式匹配 针对特定 关键词（'xqd', 'ssd', 'tv'), 包含一个特殊的 rule for “LTE” to “手机”,
        并在检查中依次回退 for “软盘驱动器” and 如果之前均未匹配, 则使用“存储卡” 找到。
        """
        如果 规范化名称 == '0':
            返回 '0'

        # 先手, 尝试匹配 特定存储器 类型使用 正则表达式。
        match = RE_MEM_TYPE_DIRECT.搜索(规范化名称)
        if 匹配:
            找到的类型 =match.group(0).lower()
            如果 找到的类型 == 'LTE':# 特殊 按要求的规则: 'LTE' 表示它是一个 '手机'
                返回 '手机'
            返回 找到的类型

        # 如果没有正则表达式匹配, 检查 '软盘驱动器' and '存储卡' 关键词。
        如果'软盘驱动器' 在 规范化名称:
            返回 '软盘驱动器'
        如果'存储卡' 在规范化名称:
            返回 '存储卡'

        返回 '0'


    def _extract_type(规范化名称: 字符串) -> 字符串:
        """
        提取 宽泛的 产品类型/产品线 描述 从规范化的 名称中。This 特征 is 包容性,
```

```
        捕获所有相关的匹配模式，例如产品系列、特定特征或颜色。返回所有找到的类型，并去重和排序。"""


        如果 规范化名称 == '0':
            返回 '0'

        # 查找所有 匹配项，使用 组合类型/系列正则表达式。
        匹配项 = RE_TYPE_SERIES_COMBINED.findall(规范化名称)
        返回 _process_matches(匹配项)


    def _extract_model(规范化名称: 字符串) ->字符串:"""

        提取简洁的型号标识符。 它应用一个 通用 正则表达式 模式 用于字母数字序列，
        可选地 包含 连字符， 并筛选出匹配项，以确保 they 至少包含 3 个字符 长。
        此举旨在 for 宽松且 全面的型号识别。
        """
        如果 规范化名称 == '0':
            返回 '0'

        匹配项 = RE_MODEL.findall(规范化名称)
        # 筛选匹配项以满足最低要求 长度 需求 (3 个或更多字符)。
        已筛选的匹配项 = [m 遍历 m 中的 匹配项 如果 len(m) >= 3]
        返回 _process_matches(filtered_matches)


    def _extract_model_long(normalized_name: 字符串) -> 字符串:
        """
        提取 更长的和 更具结构化 型号 。 它结合了 两个正则表达式 模式：
        one 针对 高度 特定， 厂商前缀 模式， and 另一个 有关更多 通用 大写
        字母数字代码通常 包括 连字符或斜杠。 这种方法 确保最大覆盖范围。
        """
        如果 规范化名称 == '0':
            返回 '0'

        提取的型号 =集合()

        # 应用 the 先手 正则表达式 针对高度 特定 长型号 模式。
        对于 m在 RE_MODEL_LONG_SPECIFIC.findall(normalized_name):
            # `re.findall` with 组可以返回元组； 提取并 add 所有非空的 组件。
            if isinstance(m, 元组):
                extracted_models.update([g.lower() 对于 g 在 m 如果 g])
            否则:
                extracted_models.add(m.lower())

        # 应用 用于匹配通用大写型号代码的第二个正则表达式。
        用于m属于RE_MODEL_LONG_GENERAL_UPPERCASE.findall(normalized_name):
            如果 isinstance(m, 元组):
                extracted_models.update([g.lower() 对于 g 在 m 如果 g])
            否则:
                extracted_models.add(m.lower())

        返回 _process_matches(列表(extracted_models))


    def _extract_model_short(规范化名称: 字符串) ->字符串: """

        提取短的， 通常 3-5个字符的型号 代码， 基于 一个预定义的 正则表达式 模式。
        这些通常是 独特的 标识符 适用于某些 产品线或变体。
        """
        如果 规范化名称 == '0':
            返回 '0'

        匹配项 = RE_MODEL_SHORT.findall(规范化名称)
        返回 _process_matches(匹配项)


    定义 _extract_features(规范化名称: 字符串) ->字符串:"""

        提取各种 技术 规格及区分 特征的 产品。
        This 包括以下元素 like USB 版本（例如， 'USB 3.0'）以HS 评分（'uhsii'）， 速度 等级（'10级'），
        and 其他特征 描述符 ， 用于全面 产品差异化。
        """
        如果 规范化名称 == '0':
            返回 '0'

        匹配项 = RE_FEATURES.findall(规范化名称)
        返回 _process_matches(匹配项)


    def _extract_item_code(规范化名称: 字符串) -> 字符串:
        """
        提取显式物品代码， 通常 制造商 零件编号， 通过模式识别
        例如括号内的数字 括号， 可能带有前缀 with “mk”。 它会清理这些 模式
        以仅返回 the 纯数字代码。 如果找到多个代码 ， the 则返回先手项。
        """
        如果规范化名称 == '0':
            返回 '0'

        匹配项 = RE_ITEM_CODE.findall(规范化名称)

        代码 = []
```

```
    # `匹配项`  here is a list 的元组,  例如, [('(mk123456)', 'mk'), ('（173473）', '')]
    对于完整匹配文本,  mk前缀组  在匹配项中:
        代码部分 = 完整匹配文本.strip('()')# 删除括号
        如果mk前缀组 并且  code_part.lower().startswith(mk_prefix_group.lower()):
            # 如果 mk前缀组  不为空,  strip 直接使用它  使用 its 准确率的长度
            代码部分 = 代码部分[长度(mk前缀组):]
        if 代码部分.isdigit():  # 确保  结果是  纯粹的 数字
            代码.append(代码部分)

    # 返回  先手提取的代码  , or '0' 如果没有 找到。
    如果代码[0]存在代码，则返回  '0'


def _extract_series(normalized_name: 字符串,  brand: 字符串) -> 字符串:
    """
    提取一个*单一的*,  most 代表  产品系列或  系列 标记。
    它遍历  通过  一个排序列表的  潜力  系列  术语（最长/最  具体的  先手）.
    尝试  to 查找  先手精确词边界  匹配规范化  名称。
    一般  或更少  判别性的  术语（例如  'USB',  'SD卡'）将被跳过。
    """
    如果 规范化名称 ==  '0':
        返回  '0'

    # 遍历  按以下顺序排列的模式  长度降序  以优先处理更具体的术语。
    对于term_pattern_str在TYPE_SERIES_RAW_PATTERNS_SORTED 中:
        # 筛选出非常通用且很少  特定  产品系列  以提高  精度。
        # 这  list 防止过度匹配  特征或通用描述符作为  '系列'。
        如果term_pattern_str.lower()  in [
            '卡',  'USB',  '存储棒',  'MB',  'GB',  'TB',  '手机',  '存储卡',  '软盘驱动器',
            '适配'  , '闪存'  , '旅行者'  , 'Ultra'  , '微型'  , '增强版' , '基础型'  , '高端'  ,
            '高  速度',  '速度',  '纤薄',  '便携',  '10级',  'UHS-I',  'USB 2.0',  'USB 3.0',
            'Type-C',  'LTE',  'XQD',  '固态硬盘',  'SDHC',  'SDXC',  'otg',  '第一代',  '第二代',  '第三代',
            '第四代',  'R-Serie',  'WEX',  'React',  'secure',  '备忘录',  '卡丁车',  '汽车',  '卡',
            '线',  '系列',  '极限',  '刀片',  '滑动',  '画布',  '风格',  '双重',  'Force',
            'Impact'  , 'a400'  , '注释'
            ]: # 扩展此内容  按需筛选列表以  提高序列准确性  vs. 常规  「类型」
            继续

        # 使用 re.search 和单词边界，确保系列标记匹配整个单词。
        如果re.search(r'\b'  + re.escape(term_pattern_str) + r'\b', 规范化名称,  标记=re.IGNORECASE):
            # 返回  先手（最具体/最长）匹配  项在  小写。
            返回 term_pattern_str.lower()

    返回  '0'# 返回  '0'  如果未找到合适的系列术语。


def _extract_pat_hb(规范化名称:  字符串) ->字符串: """

    提取  带连字符的  字母数字模式的首次出现（例如,  'uhs -i',  'Type-C'）。
    这些  模式是强烈的  技术特征或特定型号  构造的指示器。
    """
    如果 规范化名称 ==  '0':
        返回  '0'

    match = RE_PAT_HB.搜索(normalized_name)
    返回 match.group(0).lower()如果 match 否则  '0'


def _extract_hybrid(normalized_name: 字符串)  -> 字符串:
    """
    提取  '混合型'  字母数字  标识符。  这些通常是产品或型号  代码,
    由字母和数字混合组成,  and 长度至少为 5  字符  长。  示例  包括
    'DT101g2',  'USM32GQX',  用于  are key 产品阻塞。
    """
    如果 规范化名称 ==  '0':
        返回  '0'

    匹配项 = RE_HYBRID.findall(规范化名称)
    返回  _process_matches(匹配项)


def _extract_long_num(规范化名称: 字符串) ->字符串:"""

    提取所有  长度为 4 个或更多  数字  文本中的。  这些  长数字序列
    通常表示  零件编号  , 代码 , 或不带单位的容量。  匹配项为  去重后  ,
    按字典序排序,  并用单个空格连接。
    """
    如果规范化名称 ==  '0':
        返回  '0'

    匹配项 = RE_LONG_NUM.findall(规范化名称)
    # Dedupli类别,  sort 数字  ically 作为字符串ings (l按字典序  用于数字）。
    unique_sorted_nums =排序后的(列表(集合(matches)))
    返回  ' '.join(unique_排序 d_nums) if unique_sorted_nums 否则  '0'


定义 extract_features(df: pd.DataFrame)  -> pd.DataFrame:
    """
    负责协调  特征提取  流程  产品名称的 pandas 数据框  。
    它会应用一系列全面的文本规范化和  基于规则的  正则表达式提取
    to 提取结构化特征，例如  as 品牌,  容量,  型号 ,  和技术  规格。
```

```
    参数:
        df (pd.DataFrame):          输入 DataFrame,      预期包含一列                 “name”    列
                                with  产品标题（字符串                 数据类型,  文本模态）。


    返回:
        pd.DataFrame:  包含所有                                  追加了新生成的特征列。
                          新列包括:   ['规范化名称 ',                          '品牌',      '容量 ',          '内存类型 ',
                          '类型',      '型号',      '长型号 ',          '短型号 ',          '特征 ',          '物品代码 ',
                          '系列',      'pat_hb',     '混合型',     '长数字 ']。       缺失值表示为                          '0'.
    """
    # 启用      `tqdm `   进度       bars  for  pandas   `应用 `    操作,          增强          用户体验。
    tqdm.pandas()

    # 定义要提取并添加到 DataFrame 的所有特征列名称。
    特征列   =  ['规范化名称',                    '品牌',     '容量',          '内存类型',     '类型',    '型号',
                        '长型号',          '短型号',          '特征',          '物品代码',      '系列',
                        'pat_hb',  '混合型',    '长数字']

    打印('正在初始化          new  特征列使用默认字符串                          "0"...')
    # 初始化所有          new  特征列使用默认字符串                          '0'.  This  以确保
    # 所有目标列均存在并已填充,                                  even  对于没有.....的行          特征已提取。
    对于列中        特征列:
        df[col]  =   '0'# 如果列已存在则覆盖，否则创建新列。


    打印('开始多模态（文本）                    使用基于规则的特征提取和                          正则表达式方法...')

    # 步骤 1      （关键的先手步骤）:          提取       “规范化名称”。          此列用作                          主要       输入
    所有后续步骤的 #  后续          特征提取                函数,          确保          一致且                  清洗后的文本      处理。
    打印('1/14:      提取          规范化名称（文本                 清洗          & 标准化）...')
    数据框['规范化名称']  =  数据框['名称'].progress_apply(_normalize_text)

    # 优化:                创建          布尔掩码以                 识别          行，其中       `规范化名称 `          有意义          (not  '0')。
    # 后续（计算量更大的                                  ）提取操作将                          only  应用于这些相关的                          行,
    # 显著地                提升整体                  性能,                 尤其是          在大型       数据集上，使用     many  '0'  已归一化          名称。
    有意义名称掩码   =(df['规范化名称']  !=                                                  '0')
    有意义的数字名称   =   有意义名称掩码.求和()

    如果 有意义的数字名称    ==  0:
        打印("无有效          规范化          产品名称                 已找到  清洗后。                 跳过所有          后续                 提取。)
        返回 df# 提前退出                 如果没有任何内容          需要处理          超出          归一化范围。

    # 应用          其余                  特征提取                    函数仅                 应用于具有          有效      `规范化名称 `。
    #  `.loc[has_meaningful_name_mask,          “列 ']`”          高效地          仅更新          the  已筛选          行。

    # 步骤    2:  提取                 品牌      的          产品。       This  对于产品识别至关重要。
    print(f'2/14:      正在提取          来自 {num_meaningful_names} 个相关条目的品牌......')
    df.loc[有意义名称掩码,                                  '品牌']  =\
        df.loc[有意义名称掩码,                                  '规范化名称'].progress_apply(_extract_brand)

    # 步骤    3:  提取存储容量                                  （例如,  32GB,    1TB）。对.....至关重要      产品差异化。
    打印(f'3/14:      提取          容量          从 {num_meaningful_names} 个相关条目中......')
    df.loc[有意义名称掩码,                                  '容量']  =  \
        df.loc[有意义名称掩码,                                  '规范化名称'].progress_apply(提取容量)

    # 步骤    4:  提取存储器                  特定类型（例如,                 固态硬盘,    XQD）。
    打印(f'4/14:      正在提取内存类型          从 {num_meaningful_names} 个相关条目中......')
    df.loc[有意义名称掩码,                                  '内存类型']  =  \
        df.loc[有意义名称掩码,                                  '规范化名称'].progress_apply(_extract_mem_type)

    # 步骤    5:  提取          通用          产品类型或产品系列                          指示器。
    打印(f'5/14:      正在提取          type  从 {num_meaningful_names} 个相关条目中...')
    df.loc[有意义名称掩码,                                  '类型']  =  \
        df.loc[有意义名称掩码,                                  '规范化名称'].progress_apply(_extract_type)

    # 步骤    6:  提取          简洁的模型标识符。
    打印(f'6/14:      正在提取          型号（来自 {num_meaningful_names} 个相关条目......')
    df.loc[有意义名称掩码,                                  '型号']  =\
        df.loc[有意义名称掩码,                                  '规范化名称'].progress_apply(_extract_model)

    # 步骤    7:  提取更长的,          更具体的型号                          编号。
    打印(f'7/14:      正在提取          长型号          从 {num_meaningful_names} 个相关条目中......')
    df.loc[有意义名称掩码,                                  '长型号']  =  \
        df.loc[有意义名称掩码,                                  '规范化名称'].progress_apply(_extract_model_long)

    # 步骤    8:  提取短型号                          代码。
    打印(f'8/14:      提取          短型号来自 {num_meaningful_names} 个相关条目...')
    df.loc[有意义名称掩码,                                  '短型号']  =\
        df.loc[有意义名称掩码,                                  '规范化名称'].progress_apply(_extract_model_short)

    # 步骤    9:  提取技术特征（例如,                                  USB 3.0,     10 类）。
    打印(f'9/14:      正在提取          特征          来自 {num_meaningful_names} 个相关条目...')
    df.loc[有意义名称掩码,                                  '特征']  =  \
        df.loc[有意义名称掩码,                                  '规范化名称'].progress_apply(_extract_features)

    # 步骤    10:  提取明确的商品或零件代码。
    打印(f'10/14:      正在提取          物品代码      来自 {num_meaningful_names} 个相关条目......')
    df.loc[有意义名称掩码,                                  '物品代码']  =\
        df.loc[有意义名称掩码,                                  '规范化名称'].progress_apply(_extract_item_code)

    # 步骤    11:  提取                 产品系列/产品家族。                 这通常          受益于      from  对以下内容的了解      品牌,
    # 因此我们应用      在两者之间按行进行                 '规范化名称'                 and  '品牌'      列。
```

```
    打印(f'11/14:  正在提取  系列  来自 {num_meaningful_names} 个相关条目.....')
    df.loc[有意义名称掩码,  '系列'] = df.loc[有意义名称掩码].progress_apply(
        lambda 行:  _extract_series(row['规范化名称'],  行['品牌']),  轴=1
    )

    # 步骤  12:  提取带连字符的  字母数字  模式。
    打印(f'12/14:  正在从 {num_meaningful_names} 个相关条目中提取 pat_hb.....')
    df.loc[有意义名称掩码,  'pat_hb'] = \
        df.loc[有意义名称掩码,  '规范化名称'].progress_apply(_extract_pat_hb)

    # 步骤  13:  提取  '混合型'  字母数字模型标识符。
    打印(f'13/14:  正在提取  从 {num_meaningful_names} 个相关条目中提取混合型.....')
    df.loc[有意义名称掩码,  '混合型'] =\
        df.loc[有意义名称掩码,  '规范化名称'].progress_apply(_extract_hybrid)

    # 步骤  14:  提取任何  long  数字序列  (4+  数字)。
    打印(f'14/14:  正在提取  长数字  从 {num_meaningful_names} 个相关条目中...')
    df.loc[有意义名称掩码,  '长数字'] = \
        df.loc[有意义名称掩码,  '规范化名称'].progress_apply(_extract_long_num)

    打印('多功能 -模态 (文本)  特征提取已完成  成功。')
```

模型调试的数据标注 - hibug 流水线。

专家编写的流水线，添加了语义数据算子。

```
def annotation_pipeline():
    待标注样本 =skrub.var("samples_to_annotate")

    已标注 = 待标注样本.语义_提取_特征(
        nl_提示="""
        标注一组  名人  具有指定  属性的图像。  这些属性将用于  调试型号  预测
    某位名人是否  戴着  口红，出现在一张  图像中。确保  that  你的属性之间存在相关性  with  预测的潜在失败
    this  预测  任务。

        重要:  Each  属性值应  由单个词组成  或仅短语  从  列表  潜力  答案！。
        """,
        输入列=["图像"],
        名称="提取特征",
        输出列={
            "胡须":  "是否  the  人物  有  胡须? ",
            "化妆品":  "是否  the  人物化妆了吗? ",
            "性别":  "Is  the  人物  in  the  照片中的人是男性还是  女性? ",
            "发色":  "哪一种  以下  内容  发色  does  the  人物  in  the  照片  有:  金色,  棕色,  黑色,  灰色,  白色
    还是红色? ",
            "肤色":  "是否  the  人物  在  照片中  有白色  ，棕色或  黑色  皮肤? ",
            "情绪":  "哪个  of  the  以下  情绪  is  the  人物  in  the  照片  显示: 悲伤,  严肃,  平静,  快乐,  惊讶,
    中性,  愤怒,  兴奋,  沉思?",
            "年龄":  "哪个  以下  的  age  范围  is  the  人物  in  the  照片  in:  年轻,  中年,  年老?",
        },
        通过代码生成=True,
    )

    返回 已注释的
```

对应的合成算子代码。

```
导入 PyTorch
导入  pandas 作为pd
从 Transformers  导入  流水线
从Python 图像库 导入 图像
从 tqdm.auto  导入 tqdm  # 用于进度条

# 提供的设备  选择  函数
def  pick_device():
    # Apple 芯片优先使用 MPS  ，  然后使用 CUDA  ，  然后是中央处
    如果 torch.backends.mps.is_available():
        打印("正在使用  MPS  设备")
        返回  torch.device("mps")
    否则如果 PyTorch.cuda.可用():
        # 查找可用内存最多的图形处理器
        free = []
        用于 i在范围内(PyTorch.CUDA.设备数量()):
            f, t = PyTorch.cuda.mem_get_info(i)
            free.append((f,  i))
        free.sort(reverse=True)
        _,  索引 = 可用内存[0]
        打印(f"已选择  图形处理器: {idx}")
        返回  torch.device(f"cuda:{idx}")
    否则:
        打印("正在使用  中央处理器")
        返回 torch.device("中央处理器")

设备 =  选择设备()

待提取特征 = [
    # 特征  '胡须': 是/否。  与  口红相关，因为  面部毛发可能遮挡嘴部  区域。
```

```
    {'feature_name': '胡须', 'feature_prompt': '该人物是否有胡须？', 'input_columns': ['图像'], 'candidate_labels': ['是', '否']}, # 特征“化妆品”：是/否。与口红直接相关,
# 用于处理图像的批次大小定义 . 针对大型 CLIP 型号的潜在显存占用进行了调整 .
批次大小 = 16 # 对于 `大型 -patch14` 型号，较小的批次大小通常更安全，可避免内存不足错误，尤其是在较小的图形处理器或 MPS 上。

def 提取特征(df: pd.DataFrame) ->pd.DataFrame: # 本代码块中变更的理由： # 特征提取的核心逻辑沿用上一轮迭代，仍然稳健。 # 此步骤的性能进步针对所提取特
```

```
        批量图像 = [图像.open(path).convert("RGB")              对于 路径在    批量图像路径]

        for 特征配置          in 待提取特征:
            特征名称 = 特征配置['特征名称']
            候选标签 =特征配置['候选标签']

            批量预测结果 =          分类器(
                图像=批量图像,
                候选标签=候选标签                          ,
            )

            for 预测列表 在     批量预测结果:
                最佳标签 =     预测列表 [0]['标签']
                结果[特征名称].追加(最佳标签)

    for 特征名称,          特征值              在results.items():
        df[特征名称] = 特征值

    返回 df
```

A.6.4. 简化流水线代码的示例结果

我们表明，语义数据算子简化了专家编写的流水线。 表7 展示了 SEMPIPES SemOps 所减少的数据框操作数/代码行数。例如，在面向特征工程（房价）任务的 kaggle-house-b 流水线中，跨多个列的数十项数据框操作被单个 sem_gen_features 算子所取代。同样，在用于提取临床信息指导的文本特征的 micromodels 流水线中，数百行自定义索引与检索代码被单个 sem_extract_features 算子所取代。然而，“专家编写”的流水线很少由专家清理或重构，因此代码质量差异很大。智能体生成的流水线通常简洁但过于简单（这也是其性能欠佳的原因之一）；对于这些流水线，SEMPIPES 通过增加有效操作而非减少代码来提升性能。

表7：将语义数据算子应用于手写流水线后的代码简化情况。DF操作指数据框操作，LoC指代码行数。

以数据为中心的任务	流水线	代码简化
临床信息特征提取	micromodels	-250 代码行数
实体解析分块	rutgers	-77 代码行数
	sustech	-445 代码行数
特征工程（电影收入）	kaggle-movie-a	-48 DF操作
	kaggle-movie-b	-113 DF操作
特征工程（房价）	kaggle-house-a	-19 DF操作
	kaggle-house-b	-81 DF操作
特征工程（Scrabble评分）	kaggle-scrabble	-15 DF操作
数据标注	hibug	-172 代码行数
公平性数据增强	sivep	-4 DF操作

特征工程（house-prices）- kaggle-house-b 流水线。

原始专家编写的流水线代码.

```
    训练集, 测试集 = train_test_split(all_data_initial,          测试集面积=0.5,    随机种子=随机种子)

    data = pd.concat([train.iloc[:,          :-1],  测试], 轴=0)
    data = 数据.drop(columns=["标识符", "街道".      "泳池质量". "公共设施"], 轴=1)

    train_lot_median  = train.groupby("街区")["地块临街面"].变换(lambda x:x.median())
    数据["地块临街面"] = 数据.groupby("街区")["地块临街面"].变换(
        lambda x: x.fillna(train_lot_median[x.名称] if x.名称          在 train_lot_median.index 否则0)
    )
    train_msz_mode  =train.groupby("住宅类型")["土地分区"].变换(
        lambda x: x.mode()[0]    if len(x.mode())   > 0 else   "Other"
    )
    data["土地分区"] = data.groupby("住宅类型")["土地分区"].变换(
        lambda x: x.fillna(
            train_msz_mode[x.名称]
            如果x.name 在 train_msz_mode.index
            否则 (x.mode()[0]    if len(x.mode())   > 0 else   “其他”)
        )
    )
```

```
特征 = [
    "电气"          ,
    "厨房质量"      ,
    "销售类型".
    "外部材料（二）",
    "外部材料（一）",
    "小巷".
    "围栏"  ,
    "杂项特征"      ,
    "壁炉质量"      ,
    "地下室状况".
    "车库质量".
    "车库装修".
    "车库类型".
    "地下室状况",
    "地下室采光"    ,
    "地下室质量",
    "地下室装修类型2".
    "地下室装修类型1".
    "砖石饰面类型".
]
用于名称 in 特征:
    数据[名称].fillna("其他",          inplace=True)

数据["功能性"] = 数据["功能性"].fillna("typ")

零 = [
    "车库面积".
    "车库建成年份".
    "砖石贴面面积",
    "地下室半浴室"  ,
    "地下室半浴室"  ,
    "地下室全浴室"  ,
    "地下室完工面积1",
    "地下室完工面积2",
    "地下室未完工面积",
    "地下室总面积"  ,
    "车库车位数".
]
对于 名称 位于零:
    数据[名称].fillna(0, inplace=True)

data.loc[数据["住宅类型"] == 60, "住宅类型"] = 0d
 ata.loc[(data["MSSubClass"] == 20) |(d ata["MSSubClass"] == 120), "住宅类型"] = 1
data.
 ata.loc[(data["MSSubClass"] == 40) |(d ata["MSSubCl类别"] == 70)  |(数据["住宅类型"] == 80),       "住宅类型"] = 3
数据.loc[
    (数据["住宅类型"] == 50)|(数据["住宅

] = 4
data.loc[(data["住宅类型"] == 30)  | (data["住宅类型"] == 45)  | (data["住宅类型"] == 180),"住宅类型"] = 5data.loc[(data["住宅类型"] == 150),"

对象特征 =data.select_dtypes(include="object").columns

定义 哑变量(d): dummies_df =pd.DataFrame()object_features =

哑变量数据 =哑变量(数据)
数据 = 数据.drop(columns=对象特征 , axis=1)

最终数据 = pd.concat([数据 , 哑变量数据], axis=1)

训练数据 = 最终数据.iloc[: train.shape[0],:]
测试数据 = 最终数据.iloc[train.shape[0] :, :]

X = 训练数据
X.drop(columns=["售价"], inplace=True)
y = train.loc[:, "售价"]

X_test = 测试数据
y_test = 测试数据["售价"]
X_test.drop(columns=["售价"], inplace=True)

train_data.fillna(0, inplace=True)

来自 sklearn.linear_model 导入 弹性网络 , 套索交叉验证 , 岭回归, 岭回归交叉验证

型号_las_cv = 套索交叉验证(alphas=(0.0001, 0.0005, 0.001, 0.01, 0.05, 0.1, 0.3, 1, 3, 5, 10)) 型号_las_cv.fit(X
                , y)
las_cv_preds =model_las_cv.预测(test_data)
```

```
        型号_岭回归交叉验证 = 岭回归交叉验证(alphas=(0.01, 0.05, 0.1, 0.3, 1, 3, 5, 10))
        型号_岭回归交叉验证.fit(X, y)
        岭回归交叉验证预测 =型号_岭回归交叉验证.predict(X_test)

        型号_岭回归 = 岭回归(alpha=10, 求解器="auto")
        型号_岭回归.fit(X, y)
        岭回归_预测 =型号_岭回归.predict(测试数据)

        型号_en = 弹性网络(random_state=1, alpha=0.00065, 最大迭代次数=3000)
        model_en.fit(X, y)
        en_preds = 型号_en.预测(测试数据)

        导入 xgboost 作为xgb

        型号_xgb =xgb.XGBoost 回归器(
            学习率 =0.01,
            估计器数量 =3460 ,
            最大深度=3 ,
            最小子节点权重=0,
            伽马=0.
            子采样=0.7 ,
            按树列采样=0.7,
            目标函数="线性回归".
            线程数=-1 ,
            正样本权重缩放=1,
            随机种子=27,
            L1 正则化系数 =0.00006,
        )
        model_xgb.fit(X, y)
        xgb_preds =model_xgb.预测(X_test)

        自sklearn.集成模型 导入 梯度提升回归器

        model_gbr = 梯度提升回归器(
            估计器数量 =3000 ,
            学习率 =0.05,
            最大深度=4 ,
            最大特征数="sqrt" ,
            叶节点最小样本数=15 ,
            分裂最小样本数=10 ,
            损失函数="Huber损失",
            随机种子=42 ,
        )
        型号_gbr.Fit(X, y)
        gbr_preds =型号_gbr.预测(X_test)

        来自lightgbm 导入 LGBMRegressor

        型号_lgbm = LGBMRegressor(
            目标函数="回归".
            叶子节点数=4 ,
            学习率 =0.01,
            估计器数量 =5000 ,
            最大分箱数=200,
            装袋采样比例 =0.75,
            装袋频率=5,
            装袋随机种子=7,
            特征比例=0.2 ,
            特征比例随机种子=7 ,
            详细输出=-1 ,
        )
        model_lgbm.fit(X, y)
        lgbm_preds = model_lgbm.predict(X_test)

        最终预测结果 = 0.3 * 轻量级梯度提升机预测结果 + 0.3 * 梯度提升回归预测结果 + 0.1 * 极端梯度提升预测结果 + 0.3 * 岭回归交叉验证预测结果
        最终预测结果 = 最终预测结果 + 0.007 * 最终预测结果
```

采用语义数据算子的重写流水线。

```
1 类别 RegressorEnsemble(基础估计器 , 回归器混入类): 2 def __init__(self): 3 self.model_ridge_cv_ =
Rid
5
6
最大
样本
=30
1
1
1
1
1
1
1
1
1
1
```

```
21 max_depth=4,22                    =                    max_features "sqrt", 23 min_samples_leaf=15,
24
self
lear
bag
feat
self
pre
self
gbr
fina
Poo
预测
dat
能需
dat
dat
dat
reg
```

临床知情特征提取 - micromodels 流水线.

原始的专家编写流水线代码。原始流水线代码过长，无法纳入附录，但可在以下文件中查看。

- https://github.com/MichiganNLP/micromodels/blob/master/empathy/empathy_预测.py
- https://github.com/MichiganNLP/micromodels/blob/master/src/featurizers/BertFeaturizer.py
- https://github.com/MichiganNLP/micromodels/blob/master/src/micromodels/bert_query.py

使用语义数据算子重写的流水线 .

```
def sempipes_pipeline():
    帖子与回复 =skrub.var("帖子与回复")

    y = 帖子与回复["情绪反应程度"].skb.mark_as_y()
    帖子与回复 = 帖子与回复[["帖子", "回复"]].skb.mark_as_X()

    提取 = 帖子和回复。语义_提取_特征(
        自然语言提示="""
        给定 帖子和 回复 从社交 媒体 论坛, 提取某些 语言 特征（按刻度） 从 0.0 到 2.0。
        """,
        输入列 =["回复"] ,
        输出列 ={
            "情绪反应水平": """情绪 反应 级别 回复在刻度上的 从 0.0 到 2.0。
            大多数回复会有 级别 0。
            包含以下内容的回复 该文本 “我会一直支持 你们，伙计。” 具有1级。
            A 包含以下内容的回复 该文本 “我同意他的说法” has 1级。
            A 包含 文本 “只管继续练习, 永远不要放弃。” has 1级。
            包含以下内容的回复 文本 “我很抱歉你 有这种感受。” 级别为 2级。
            A 包含 回复 “天哪 我太能感同身受了。” has 2级。
            A 包含 文本 “真的 抱歉，正在 经历这一切。” has 2级。""",
            "sp_interpretation_level": """该 解释 级别 回复的刻度 范围为 0.0 到 2.0.
            大多数回复的级别为 0。
            一条包含 文本 “人们问我为什么 我大部分时间都在走神。 我不是在走神, 我只是在机器人
        这样我就不必 依附于任何事物。 它能帮我抑制 抑郁的 想法 但每次 夜晚, 这一切都只是
        流程 回来了。 我讨厌它。' has 1级。
            一条回复，其中包含 文本 “我跳过了 今天的一节课 因为我从中学不到什么 类别之外, 但我忘了
         有5%的 成绩 用于课堂活动, 所以我会 可能 不要跳过任何一个 更多类别的 has 1级。
            包含以下内容的回复 文本 “实际上 我发现 自己抑郁时，洗澡的时间会长得多。 有时一天会洗两次
        。 这是唯一的 时间，我会觉得 放松。' 具有1级。
            包含以下回复 文本 'No， 这就是我正在做的事情，但它不起作用。' 具有2级。
            一条包含 这段文本 “我不再 处理 自己的问题了..... 它们只会编译。 我停止了 对他们念念不忘，或者
         处理 他们，以任何方式。我无法控制我的 生活。我只是 看着我的问题 从我的沙发上.....' 有 2级。
            一条回复，内容是 这段文本 '我理解你的感受 。' 级别为2级。
            """,
            "探索水平": """探索 水平 回复的刻度 范围为0.0至2.0。
            Most 回复将具有 级别 0。
            包含以下内容的回复 文本 “我们今天能做些什么 来提供帮助？” has 1级。
            包含以下内容的回复 文本 '发生了什么？' has 1级。
            包含以下内容的回复 这段文本 “你凭什么觉得 你是个糟糕的 人 ？” has 1级。
            一个回复包含 文本 “是什么让你说 这些话？” 具有2级。
            A 回复包含 这段文本 “你觉得 是什么 引发了这些令人不安的 想法？” has 2级。
            一条包含 文本 “你有没有 这样的朋友 不会总是对你阿谀奉承 或是.....
        治疗师？' has 2级。
            """,
        },
        名称="响应特征" ,
        通过代码生成=True,
    )

    X = 已提取[["情绪反应水平" , "sp_interpretation_level" , "探索水平"]]

    emo_ebm = 可解释提升分类器(
        特征名称=["情绪反应水平" , "sp_interpretation_level" , "探索水平"]
    )
    返回 X.skb.应用(emo_ebm , y=y)
```

A.7. 第 5.2 节“语义数据算子的表现与专用方法相当或更优”的详细设置

我们提供了更多 关于这组实验的细节。如正文所述，我们使用谷歌 的 gemini-flash-2.5 来运行 SEMPIPES 和基线方法。

关于特征工程实验设置的更多细节。与 CAFE 对比的实验代码可在此处获取：https://github.com/deem-data/sempipes/tree/v1/experiments/caafe。数据集、元数据以及 CAAFE 预先计算的特征工程代码取自https://github.com/noahho/CAAFE。对于 SEMPIPES，我们在训练划分上运行优化器 24 次迭代，以合成特征工程代码，并在与 CAAFE 相同的测试划分上进行评估。生成的运行日志和优化后的特征工程代码可在此处获取：https://github.com/deem-data/sempipes/tree/v1/experiments/caafe/sempipes。

缺失值填补实验设置的更多详情。实验代码可在此处获取：https://github.com/deem-data/sempipes/tree/v1/experiments/missing_值。我们将结果与（Narayan 等人，2022年）中的两个基准任务进行比较：填补餐厅数据集（Restaurant）中的城市列，以及电子产品数据集（Buy）中的制造商列。我们将结果与（Narayan 等人，2022年）代码仓库中的提示方案进行比较。该方案将表格行序列化为文本，并使用十个手工挑选的少样本示例。我们测量填补值的准确率。

多模态特征提取实验的更多详情。我们评估语义_提取_特征算子 SEMPIPES 从文本、图像和音频数据中提取特征。实验代码可在此处获取：https://github.com/deem-data/sempipes/tree/v1/experiments/feature_提取。

实验设置。我们评估三种模态。对于文本，我们使用法律金融条款数据集（Alrashidan，2022年），并预测条款类型。对于图像，我们使用胸部X光片（Kermany等人，2018年），并预测肺炎。对于音频，我们使用 ESC-50（Piczak，2015年），并预测环境声音类别。为控制成本，我们在每个数据集的 1,000 和 10,000 个项目上进行评估。

我们将 SEMPIPES 与 Palimpzest（Liu 等人，2025年）和 LOTUS（Patel 等人，2025年）进行比较；这两种系统都支持对分析型查询进行基于大语言模型的处理。在本实验中，Palimpzest 和 LOTUS 将大语言模型直接应用于每个项目，而 SEMPIPES 则合成可执行代码，并在图形处理器上本地运行。SEMPIPES sem_extract_features 也支持逐项大语言模型模式。所有系统均使用谷歌的 gemini-2.5-flash。

表 8：从法律文本（Alrashidan，2022年）、X光成像（Kermany等人，2018年）和环境声音（Piczak，2015年）数据集中提取多模态数据特征的准确率。我们从每个数据集中抽取 1,000 个数据点。成本为五次运行的总成本。

模态	Palimpzest 准确率	Palimpzest Cost	LOTUS 准确率	LOTUS Cost	SemPipes（代码） 准确率	SemPipes（代码） Cost	SemPipes（大语言模型） 准确率	SemPipes（大语言模型） Cost
Text	0.813 ± 0.004	*$1.50*	0.827 ± 0.000	*$0.866*	0.711 ± 0.021	*$0.25*	**0.829** ± 0.002	*$2.87*
图像	0.570 ± 0.000	*$3.99*	0.634 ± 0.000	*$3.13*	**0.756** ± 0.000	*$0.58*	0.699 ± 0.000	*$6.54*
音频	0.681 ± 0.000	*$2.09*	—		**0.883** ± 0.000	*$0.09*	0.628 ± 0.000	*$8.44*

表9：从法律文本（Alrashidan，2022年）、X光成像（Kermany等人，2018年）以及环境（Piczak，2015年）声音数据集提取多模态数据的准确率。我们从每个数据集中抽取10,000个数据点。成本报告为五次运行的总成本。

模态	Palimpzest 准确率	Palimpzest Cost	LOTUS 准确率	LOTUS Cost	SemPipes（代码） 准确率	SemPipes（代码） Cost	SemPipes（大语言模型） 准确率	SemPipes（大语言模型） Cost
Text	0.662 ± 0.003	*$15.37*	0.822 ± 0.000	*$8.65*	0.633 ± 0.000	*$0.16*	**0.839** ± 0.040	*$25.76*
图像	0.608 ± 0.000	*$39.89*	0.638 ± 0.000	*$31.37*	**0.745** ± 0.002	*$0.62*	0.573 ± 0.000	*$41.37*
音频	0.664 ± 0.000	*$21.19*	—		**0.894** ± 0.000	*$0.11*	0.591 ± 0.040	*$39.04*

A.8. 第 5.3 节详情："语义数据算子中不同大语言模型对代码合成的优化效果"

我们设计了一个优化器基准测试，其中包含多个语义流水线，用于在来自 skrub 项目的示例数据集上执行分类任务。对于每条流水线，我们在不同搜索策略和用于代码生成的大语言模型配置下，基于训练数据优化算子代码。具体而言，我们评估了 OpenAI 的 GPT-4.1、谷歌的 Gemini-2.5-Flash 和阿里巴巴的 Qwen-Coder-2.5 (32B) 作为代码合成器，并将其与多种搜索策略结合。每条流水线均进行五次独立运行，每次运行的优化预算为 36 步。每次运行得到的最佳算子状态都会在留出测试集上进行评估。我们报告相对于第0步的平均归一化改进，第0步对应于空算子状态（即语义数据算子处于停用状态）。归一化按流水线分别进行，使用最小–最大归一化分数。这些实验的代码和日志可在 https://github.com/deem-data/sempipes/tree/v1/experiments/colopro 获取。

基准测试流水线。该实验由在五种不同流水线上进行的运行组成，这些流水线采用多种语义数据算子，用于解决分类问题；示例数据来自 skrub 项目和 UCI 数据集存储库。

- `midwest` - 分类流水线，包含 语义_世代_特征 ，基于 2,000 条记录，数据来自 https://skrub-data.org/stable/reference/generated/skrub.datasets.fetch_midwest_survey.html#skrub.datasets.fetch_midwest_survey

•交通 - 分类流水线，包含 语义_世代_特征 ，基于 10,000 条记录，数据来自 https://skrub-data.org/stable/reference/generated/skrub.datasets.fetch_交通_violations.html#skrub.datasets.fetch_交通_violations

• 欺诈 - 分类流水线，使用sem_agg_特征 在 4,000 条记录上，来自 https://skrub-data.org/stable/reference/generated/skrub.datasets.fetch_credit_fraud.html#skrub.datasets.fetch_credit_fraud

• 客户流失 - 分类流水线，使用sem_agg_特征 在 10,000 条记录上，来自 https://archive.ics.uci.edu/dataset/352/online+retail

• 推文 - 分类流水线，使用 sem_extract_特征 在 600 条记录上，来自 https://skrub-data.org/stable/reference/generated/skrub.datasets.fetch_toxicity.html#skrub.datasets.fetch_toxicity

树搜索策略。针对每条流水线和代码合成大语言模型评估以下树搜索策略。

• 蒙特卡洛树搜索 (Browne 等人, 2012)，其中待演化的搜索节点 i 通过按照“树的上置信界”策略 $\frac{w_i}{n_i} + c \cdot \sqrt{\ln {}_{n_i}{}^{N}}$ 遍历树来选择。此处 n_i 表示节点 i 被访问的次数， w_i 指遍历 n_i 所得的效用总和， N 是父节点 i 的访问次数， c 是用于权衡利用与探索的超参数。我们在实验中设置 $c = 0.5$，当根节点拥有三个子节点时视为已完全扩展，当中间节点拥有两个子节点时视为已完全扩展。

• 截断选择 (Toledo 等人， 2025年），其中要演化的搜索节点 i 从种群中以概率 k选择得分最高的 个搜索节点之一 $\mu_i / \sum_{j=0}^{k} \mu_j$，其中 μ 表示留出数据上的效用。我们在实验中设置 $k = 6$。

• 贪心树搜索 (Jiang 等人， 2025a；Chan 等人， 2025年），该方法贪心地选择当前最佳搜索节点进行演化，同时维护固定数量的 k 个尚未探索的草稿节点。我们在实验中设置 $k = 2$。每个草稿节点都包含截至目前在搜索树中观察到的三个最佳状态，作为灵感。

• 随机搜索，这是一种简单的基线方法，会反复从头合成代码，而不维护搜索树。

其他发现。除 第 5.3 节中的结果外，我们还发现，开源模型 Qwen-Coder-2.5 尽管参数规模相对较小（32B），却表现出令人惊讶的竞争力，足以媲美参数规模大得多的大语言模型。这表明，我们的语义算子设计有助于以一种降低大语言模型搜索负担的方式构建合成空间，从而使较小的模型也能有效发挥作用。