AutoML-Agent: A multi-Agent LLm Framework for Full-Pipeline AutoML

ICML'25

Patara Trirat[1], Wonyong Jeong[1], Sung Ju Hwang[2]

Presenter: Jinhan Xin

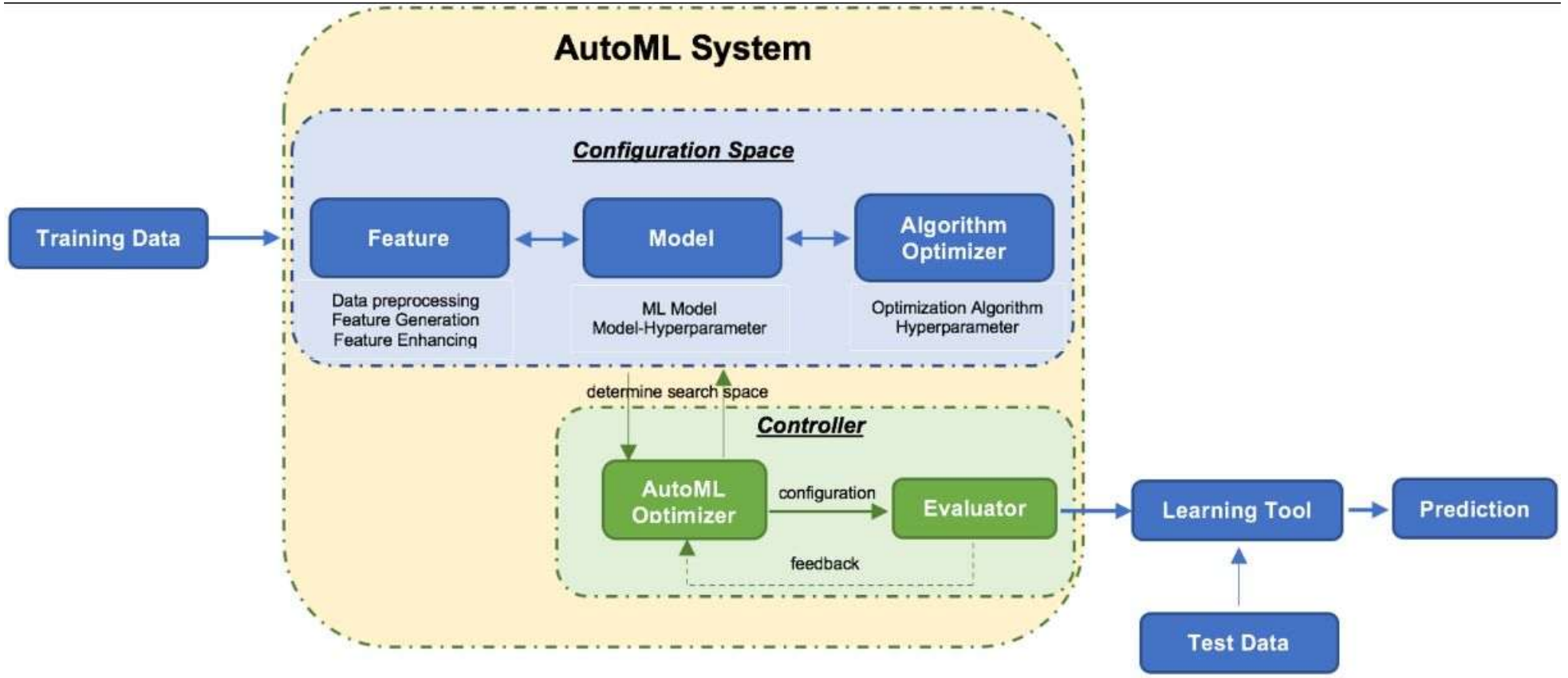
2025.09.18

内容

- 背景
- 系统设计
- 评估实验
- 思考

背景

自动化机器学习（AutoML）通过自动化人工智能开发流程中的每个环节（如特征工程、模型选择和超参数优化（HPO）），来降低技术门槛和减少人工。



背景

当前的 AutoML 系统通常需要编程专业知识来配置复杂的工具和资源，这对技能和知识有限的大量用户造成使用障碍。

之后建议使用具有LLM的自然语言界面来进行机器学习（ML）和数据科学（DS）任务。

然而，这些先前的基于 LLM 的 AutoML 框架，不仅只考虑了有限数量的任务，而且成本高、计算缓慢：

- 要么仅针对流程中的一个环节（特征工程、模型选择）、要么针对特定的一组下游任务（自然语言处理、计算机视觉）
- 忽视LLM通过实际训练候选模型来寻找有前途模型的固有能力

背景

作者认为真正实用的AutoML系统，它应该执行端到端的流程，同时考虑数据方面和模型方面。这是因为一个方面中的过程可能会影响另一个方面中后续的过程。

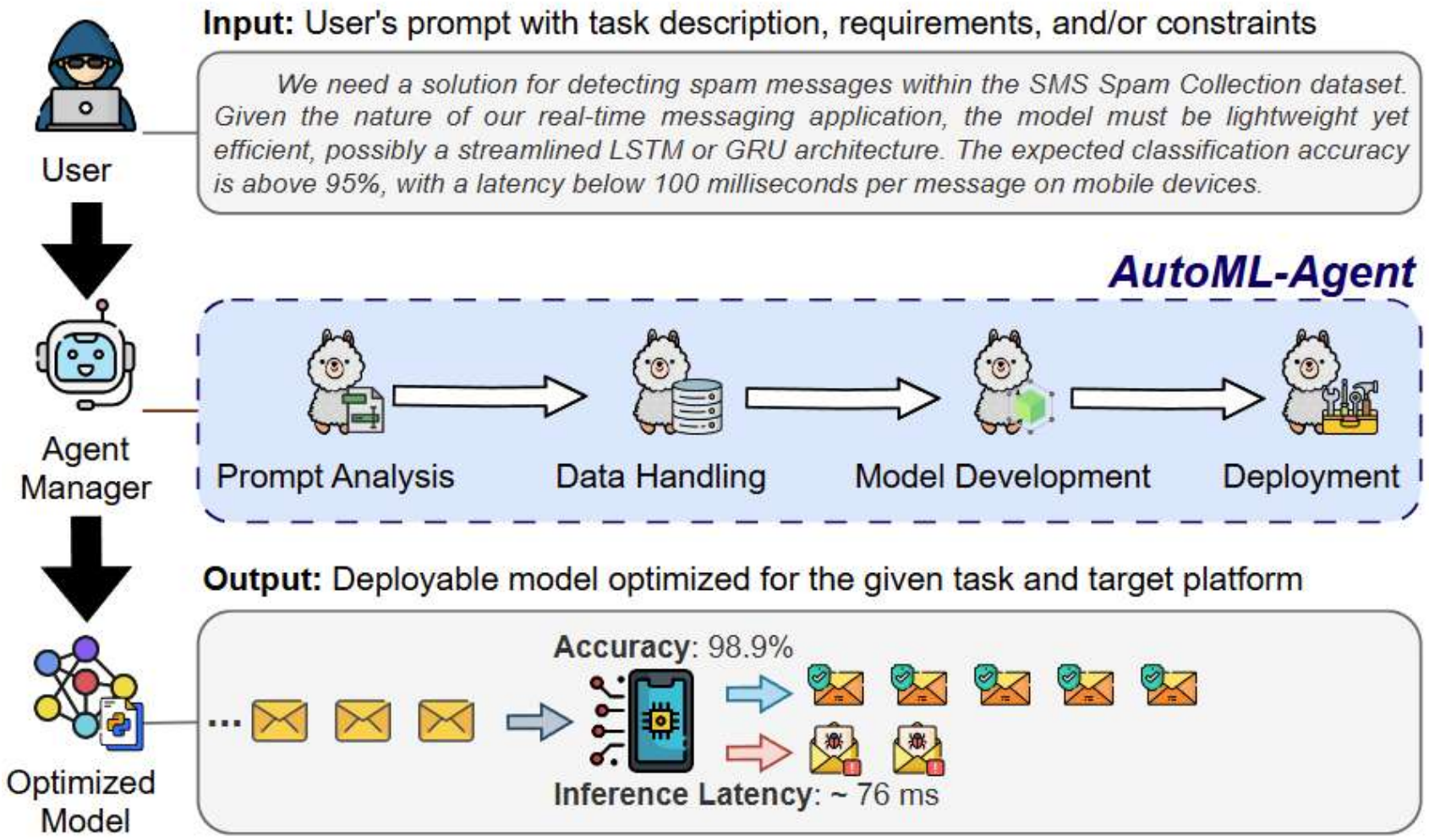
同时，AutoML 框架应该是计算效率高的，使用策略来最小化搜索过程中的计算开销。

构建此类框架时存在两个主要挑战：

- 规划任务的高复杂性：整个 AutoML 流程引入了额外的复杂性，这主要是由于流程中各步骤之间的相互依赖性。

- 准确实施中的挑战：使用 LLMs 自动生成完整的机器学习流程可能导致幻觉问题。

背景



为应对这些挑战，提出了一种多智能体框架 AutoML-Agent，用于从数据与模型搜索到评估的全流程 AutoML 。

如图所示，AutoML-Agent 接受用户的任务描述，并协调多个专业智能体协同识别最优的机器学习流程，最终输出一个部署就绪的模型及其推理端点。

背景（贡献）

- 提出了全流程的AutoML系统

- 引入基于检索的规划、角色特定的计划分解和基于提示的计划执行，解决了全流程 AutoML 中规划问题的复杂性挑战。

- 集成了基于结构的提示解析和多阶段验证，以确保实际代码实施之前所得到的解决方案和指令的质量，从而提高整体性能。

背景（相关工作）

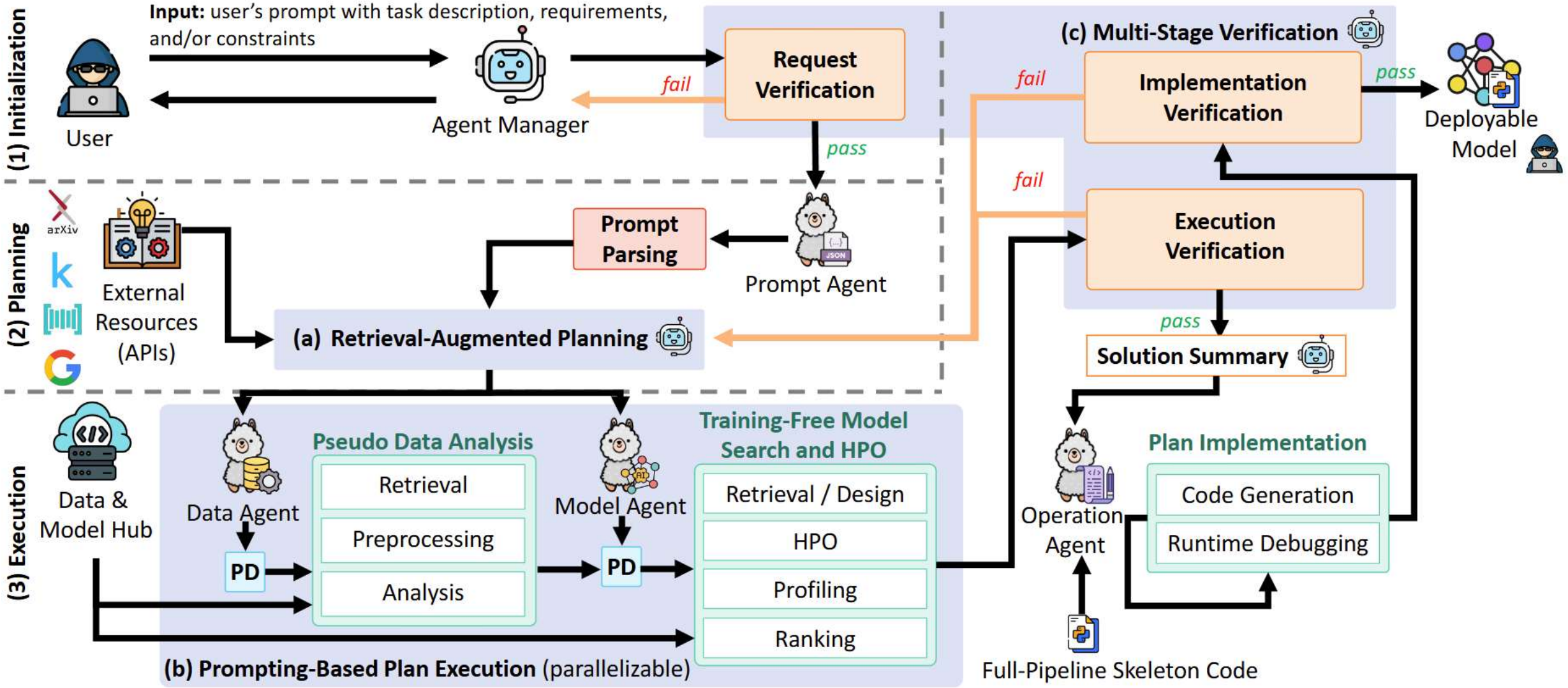
Table 1. Comparison between *AutoML-Agent* and existing LLM-based frameworks.

Framework	Key Functionality					
	Planning	**Verification**	**Full Pipeline**	**Task-Agnostic**	**Training-Free Search**	**With Retrieval**
AutoML-GPT (Zhang et al., 2023)	×	×	×	✓	✓	×
Prompt2Model (Viswanathan et al., 2023)	×	×	✓	×	×	✓
HuggingGPT (Shen et al., 2023)	✓	×	×	✓	✓	✓
CAAFE (Hollmann et al., 2023b)	×	✓	×	×	×	×
MLCopilot (Zhang et al., 2024a)	×	×	×	✓	✓	×
AgentHPO (Liu et al., 2025)	✓	✓	×	✓	×	×
Data Interpreter (Hong et al., 2024a)	✓	✓	×	✓	×	×
DS-Agent (Guo et al., 2024a)	✓	✓	×	✓	×	✓
SELA (Chi et al., 2024)	✓	✓	×	✓	×	×
Agent K (Grosnit et al., 2024)	✓	✓	×	✓	×	✓
AutoMMLab (Yang et al., 2025)	×	✓	✓	×	×	×
AutoML-Agent (Ours)	✓	✓	✓	✓	✓	✓

内容

- 背景
- 系统设计
- 评估实验
- 思考

系统设计（概览）

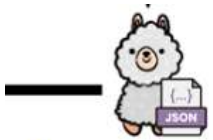

(1) Initialization
Input: user's prompt with task description, requirements, and/or constraints
User
Agent Manager
Request Verification
fail
pass
(c) Multi-Stage Verification
Implementation Verification
pass
Deployable Model
fail
(2) Planning
arXiv
External Resources (APIs)
Prompt Parsing
Prompt Agent
Execution Verification
fail
(a) Retrieval-Augmented Planning
pass
Solution Summary
(3) Execution
Data & Model Hub
Data Agent
Pseudo Data Analysis
Retrieval
Preprocessing
Analysis
PD
Model Agent
PD
Training-Free Model Search and HPO
Retrieval / Design
HPO
Profiling
Ranking
Operation Agent
Plan Implementation
Code Generation
Runtime Debugging
(b) Prompting-Based Plan Execution (parallelizable)
Full-Pipeline Skeleton Code

系统设计（概览）

Agents:

Agent Manager ($\mathcal{A}_{mgr}$) acts

协调搜索过程、与用户交互、制定全局计划、分配任务给相应的智能体、通过反馈验证执行结果、跟踪系统进度。



Prompt Agent ($\mathcal{A}_p$)

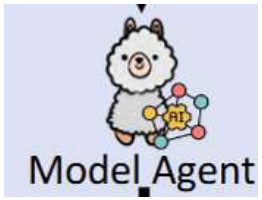
负责解析用户指令并进行指令微调的 LLM，将其解析为具有预定义键的标准化 JSON 对象。

Data Agent ($\mathcal{A}_d$)

是一个用于执行与数据操作和分析相关任务的 LLM。

系统设计（概览）

Agents:



Model Agent $(\mathcal{A}_m)$

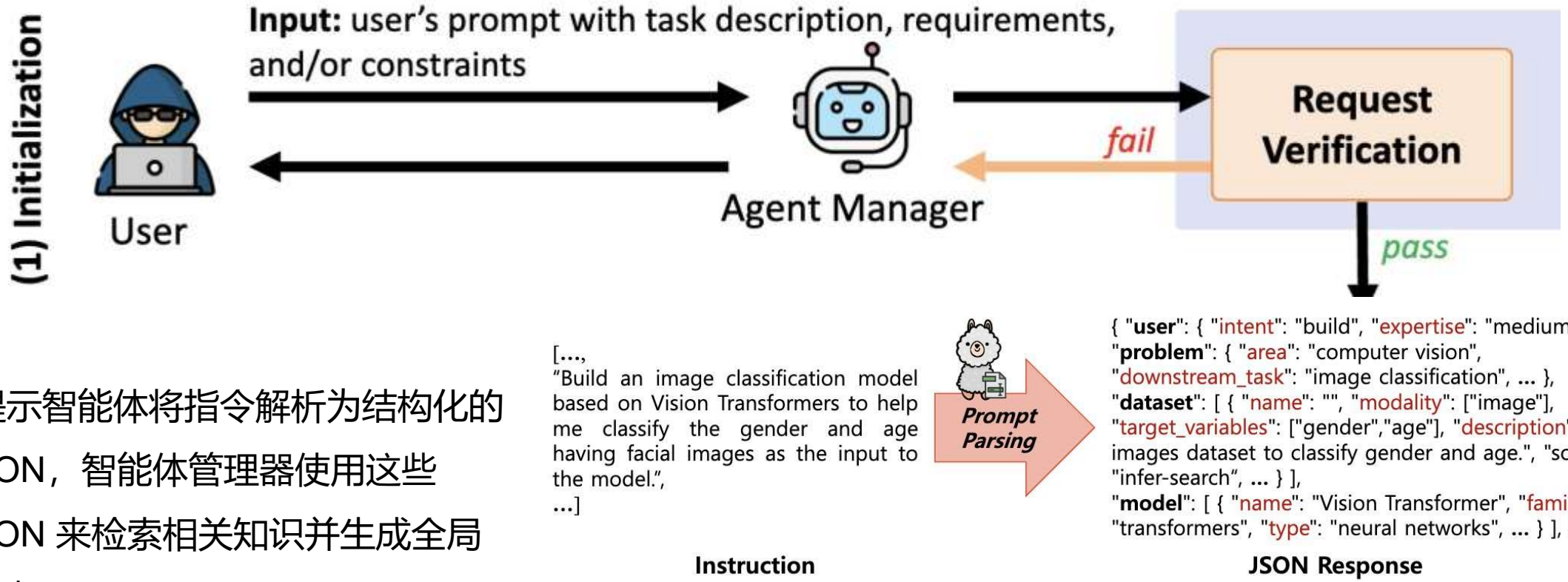
是一个用于执行模型搜索、超参数优化、模型分析和候选排名任务的 LLM。

Operation Agent $(\mathcal{A}_o)$

是一个用于实施数据智能体和模型智能体找到的、通过代理管理器验证的解决方案的 LLM。

系统设计

智能体管理器首先检查用户指令：如果有效，它将其转发给提示代理；而如果无效，它会要求用户重新说明。



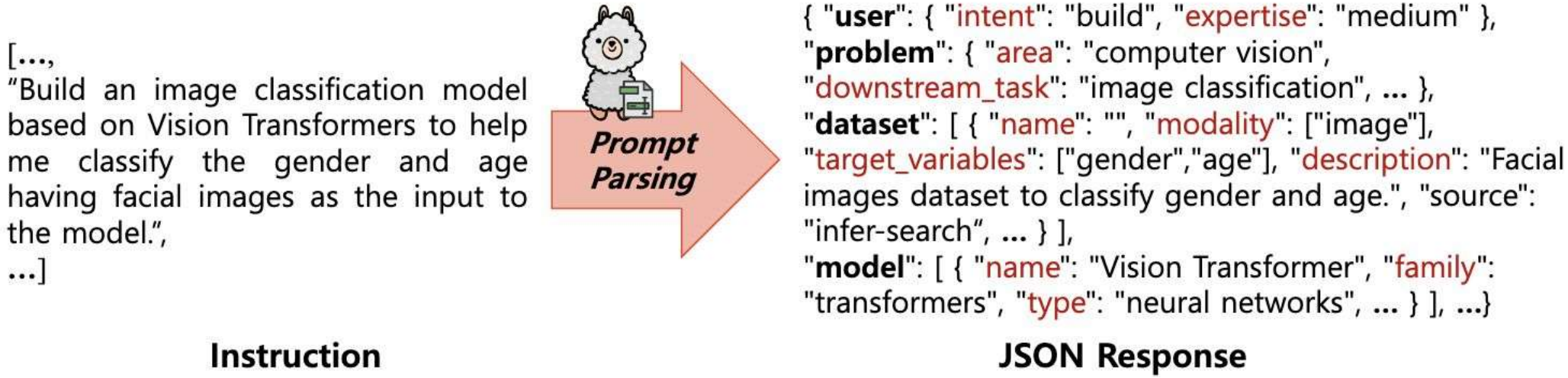
提示智能体将指令解析为结构化的 JSON，智能体管理器使用这些 JSON 来检索相关知识并生成全局计划。

系统设计

指令数据生成与提示解析

首先手动创建一组高质量的种子指令，然后自动生成一个更大的指令数据集 $D = \{(I_i, R_i)\}_{i=1}^{N}$

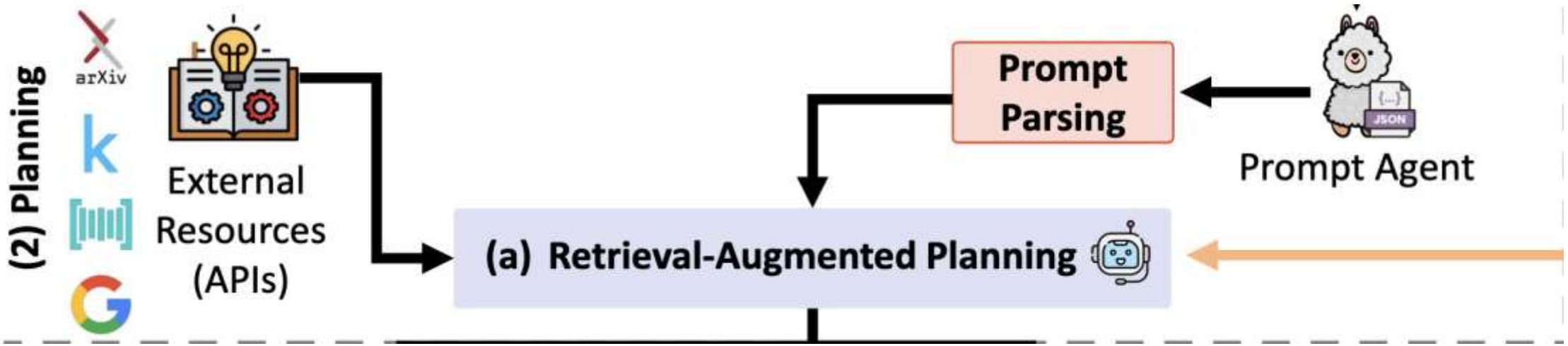
包含 N 个指令-响应对，使用 扩展的 JSON 格式来表示响应 R，然后使用指令数据集D训练LLM，用作 $\mathcal{A}_p$



这些键从各个角度提供了上下文信息，用于生成高质量的 AutoML 流程。

系统设计

提出了一种增强检索的规划策略，通过访问外部知识源生成多个计划，以探索最佳解决方案。



每个全局计划随后被细分为更小的子任务（例如，数据加载、模型搜索、超参数优化），并分配给数据智能体和模型智能体，这些智能体通过提示模拟执行。

系统设计

检索增强规划

该策略通过以下步骤生成一组计划 $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_P\}$

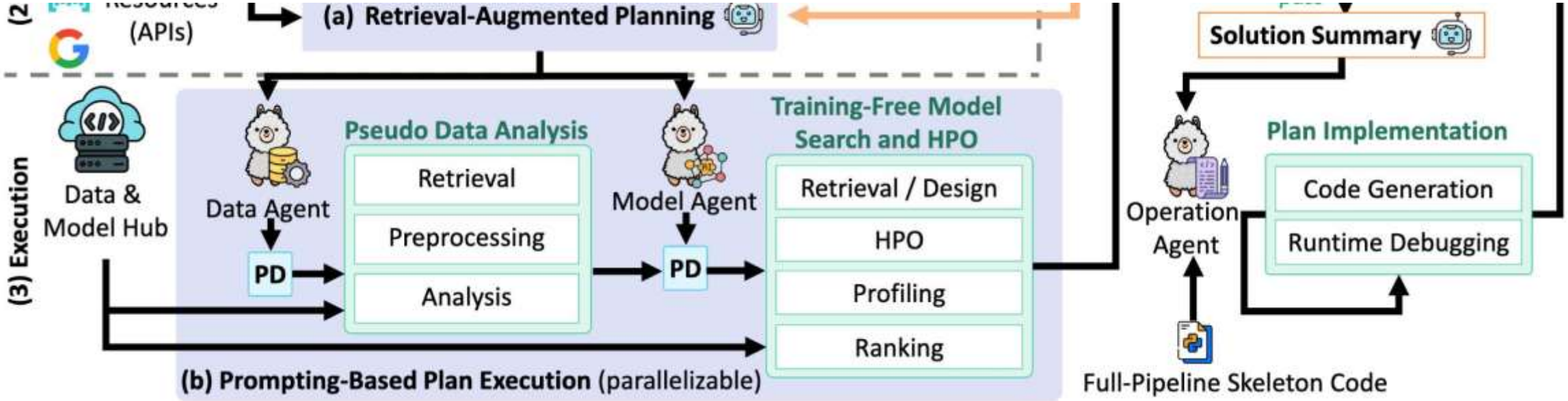
1.**知识检索与总结：**使用解析需求R，通过API调用获取相关知识和洞见（Web搜索、论文总结）。

2.**计划生成：**智能体管理器基于检索结果和LLM内部知识，生成P = {P_1, …, P_p}。每个P_i针对不同场景（如不同数据处理策略或模型架构），是完整的AutoML流程计划。

3.**独立性设计：**智能体管理器独立设计每个计划，使后续子步骤（如数据预处理、神经网络设计）可并行执行。

系统设计

数据智能体执行数据集检索、预处理和分析；模型智能体寻找合适的模型，进行超参数优化（HPO）、模型分析和排名。



在模拟过程中，智能体管理器选择的最佳计划随后由操作代理执行，该代理编写实际代码并进行运行时调试。

系统设计

基于提示的计划执行

计划分解：将原始计划 $P_i \in P$拆分为一组较小的子任务 s_i，这些子任务与代理的角色和专长相关，以提高大型语言模型（LLMs）在解决和执行计划时的有效性。具体而言，子任务 $s_i^d = PD(R, A_d, P_i)$由数据智能体A_d生成，包含针对计划 P_i的子任务。然后，智能体根据分解后的计划执行，以满足用户需求 R，而不是直接执行原始冗长的计划。(子任务 s_i^m的定义依赖于数据代理的输出)。

伪数据分析：在 AutoML-Agent 中，数据智能体A_d处理子任务 s_i^d，包括数据集的检索、预处理、增强和分析。在数据检索阶段，如果用户未直接提供数据集，会通过 API 调用（如 HuggingFace 和 Kaggle）根据数据集名称或描述搜索潜在数据集。找到数据集后，会用其元数据增强提示；若未找到，则依赖 LLM 的内在知识。随后，提示 A_d按数据集特性和用户需求 R执行 s_i^d。子任务的总结结果 O_i^d转发给 A_m，其中 $O_i^d = A_d(s_i^d)$。

系统设计

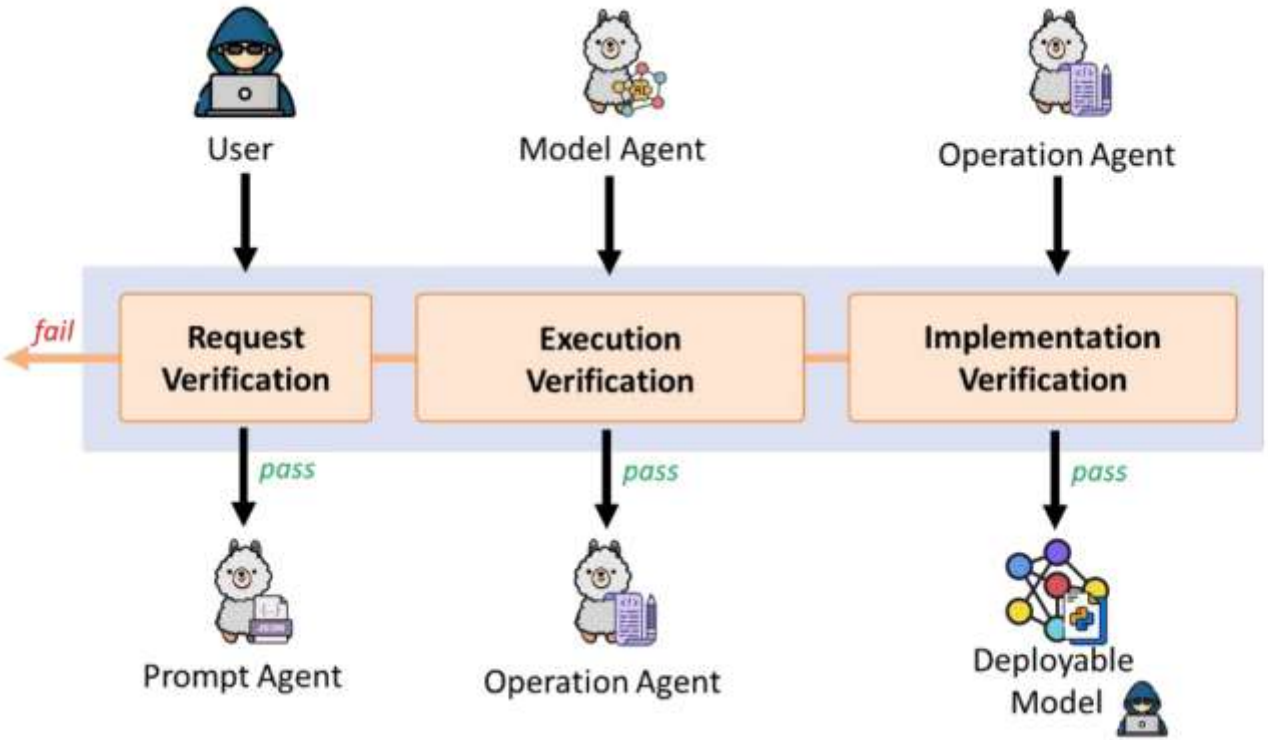
基于提示的计划执行

无训练模型搜索和 HPO：类似 A_d，A_m 通过 API 调用完成子任务 s_i^m，而非直接执行代码。与 A_d不同，A_m 的计划分解会整合 A_d的结果，从而识别预处理数据集的特性，即 $s_i^m = PD(R, A_m, P_i, O_i^d)$。$A_m$ 执行 s_i^m，包括模型检索、运行 HPO（超参数优化）及总结结果，这些结果包括预期数值性能指标（如准确率和错误率）以及模型复杂性因素（如模型大小和推理时间）。还提示代理返回前 k个最有前景的模型，形式上 $O_i^m = A_m(s_i^m)$。

计划执行：为提高 A_o在代码生成中的效率，A_{mgr} 首先验证从 A_d和 A_m执行结果 $O = \{O_i^d O_i^m\}_{i=1}^P$。$A_{mgr}$ 选择最佳结果 $O^* \in O$，并生成指令 I^*给 A_o编写实际代码。形式上，$M^* = A_o(I^*)$，其中 M^*是可部署的模型。

系统设计

为了确保生成代码的可靠性，我们还加入了多阶段验证流程。



这个过程包括中间检查和验证，引导代理完善输出并遵循用户要求。

系统设计

多阶段验证

包含三个验证步骤以确保其准确性和有效性：请求验证、执行验证和实施验证

请求验证：根据用户指令的清晰度来判断其是否与执行机器学习任务相关，并是否充分。如果指令不足以进入规划阶段，智能体管理器 会请求补充信息，从而促进多轮对话。

执行验证：A_{mgr}验证由 A_d和 A_m生成的结果集 O是否满足用户需求。如果结果令人满意，系统会选择最优解决方案进行实施，从而减少搜索过程中的计算开销，将资源集中分配给最有前景的方案。

实施验证：这一阶段与 执行验证类似，但重点在于验证由 A_o执行并编译的代码生成的结果。如果结果符合用户需求，A_{mgr} 提供模型和部署端点给用户；若不满足，则系统会记录失败，并转入计划调整阶段。

内容

- 背景
- 系统设计
- 评估实验
- 思考

实验设置

选择了七个下游任务，涉及五种不同数据模态，包括图像、文本、表格、图和时间序列

Table 2. Summary of downstream tasks and datasets.

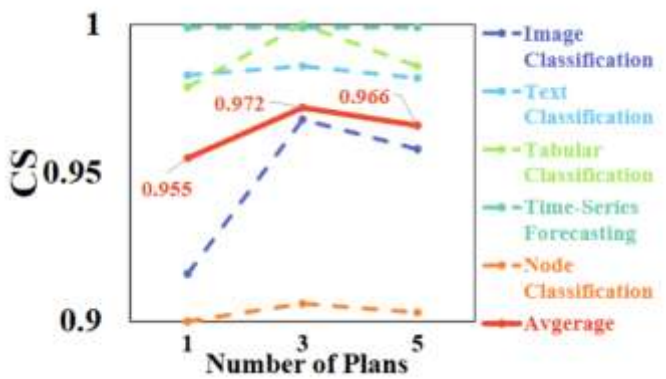
Data Modality	Downstream Task	Dataset Name	Evaluation Metric
Image (Computer Vision)	Image Classification	Butterfly Image Shopee-IET	Accuracy
Text (NLP)	Text Classification	Ecommerce Text Textual Entailment	Accuracy
Tabular (Classic ML)	Tabular Classification	Banana Quality Software Defects	F1
	Tabular Regression	Crab Age Crop Price	RMSLE
	Tabular Clustering	Smoker Status Student Performance	RI
Time Series (Time Series Analysis)	Time-Series Forecasting	Weather Electricity	RMSLE
Graph (Graph Mining)	Node Classification	Cora Citeseer	Accuracy

评估指标：采用了综合得分（CS），同时评估代码生成的成功率（SR）和构建流程的归一化性能得分（NPS）。公式为 $CS = 0.5 \times SR + 0.5 \times NPS$，其中 $NPS = \frac{1}{1+s}$，s 为基于损失的性能得分，例如 RMSLE。

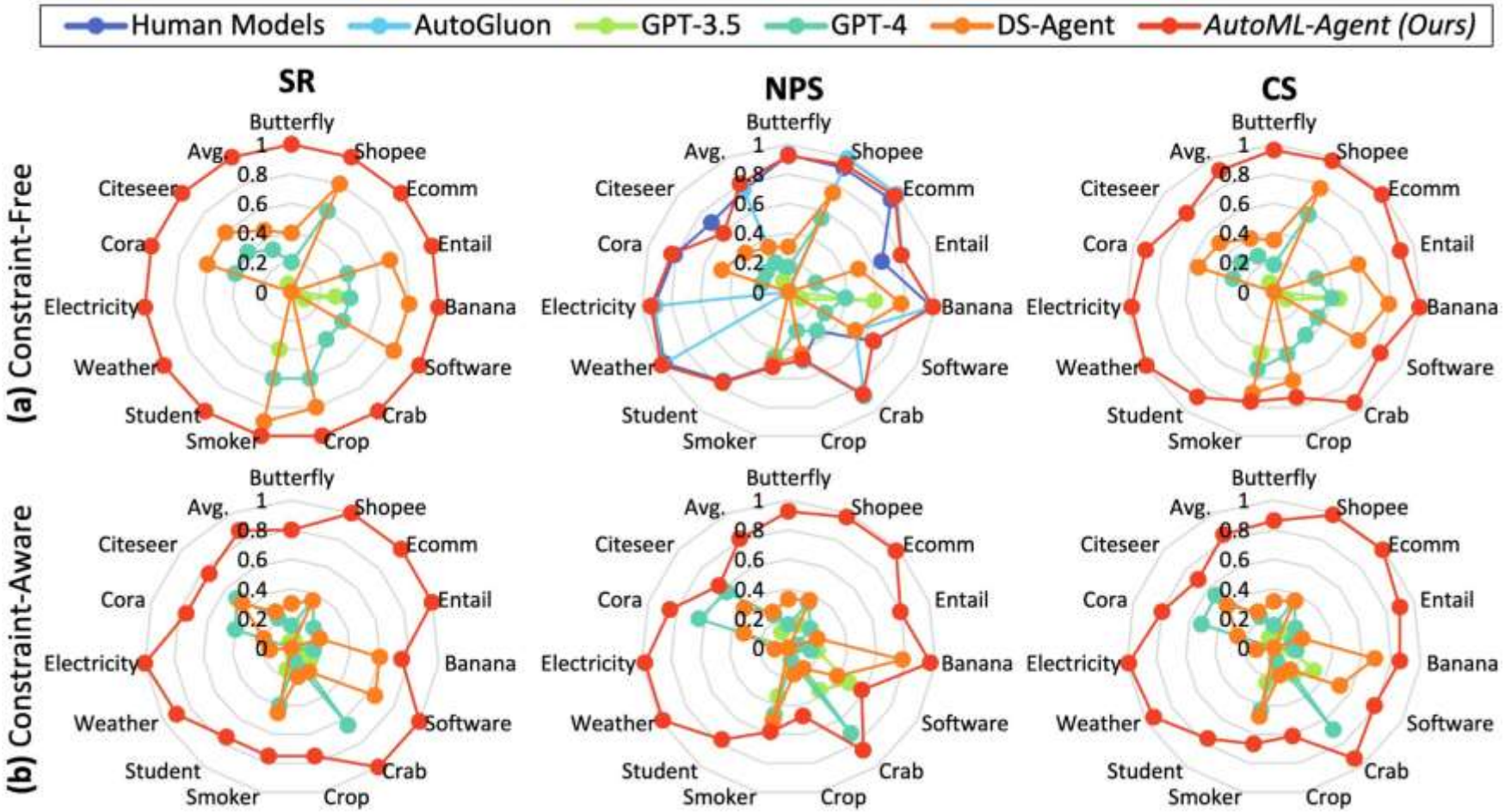
实验设置

Baseline：由于这是基于 LLM 的全新全流程AutoML 框架，缺乏直接基线。实验将 AutoML-Agent 与任务特定的人工设计模型及以下基线比较AutoML、DS-Agent 、通用 LLM（GPT-3.5, GPT-4），后者使用零样本提示。

实现细节：除 A_p使用 Mixtral-8x7B 外，所有代理和 LLM 基线均采用 GPT-4 作为骨干模型，以确保公平性能评估。为指令调整 A_p，通过自动生成约 2.3K 指令-响应对并使用 LoRA 微调模型。对于 RAP，设置计划数 $P = 3$，候选模型数 $k = 3$。



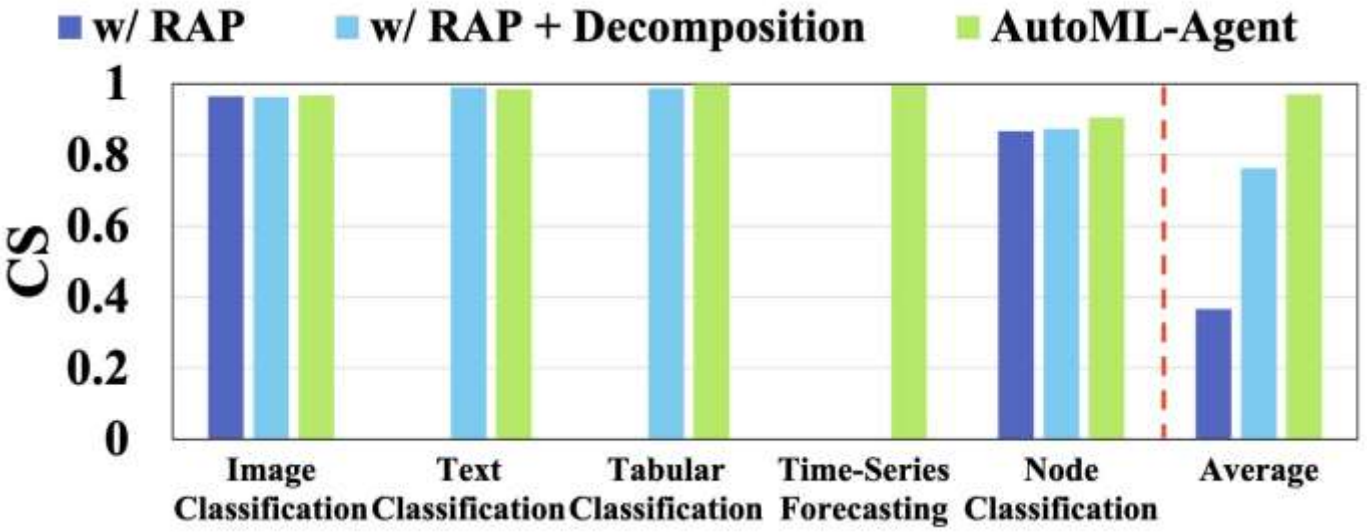
结果评估

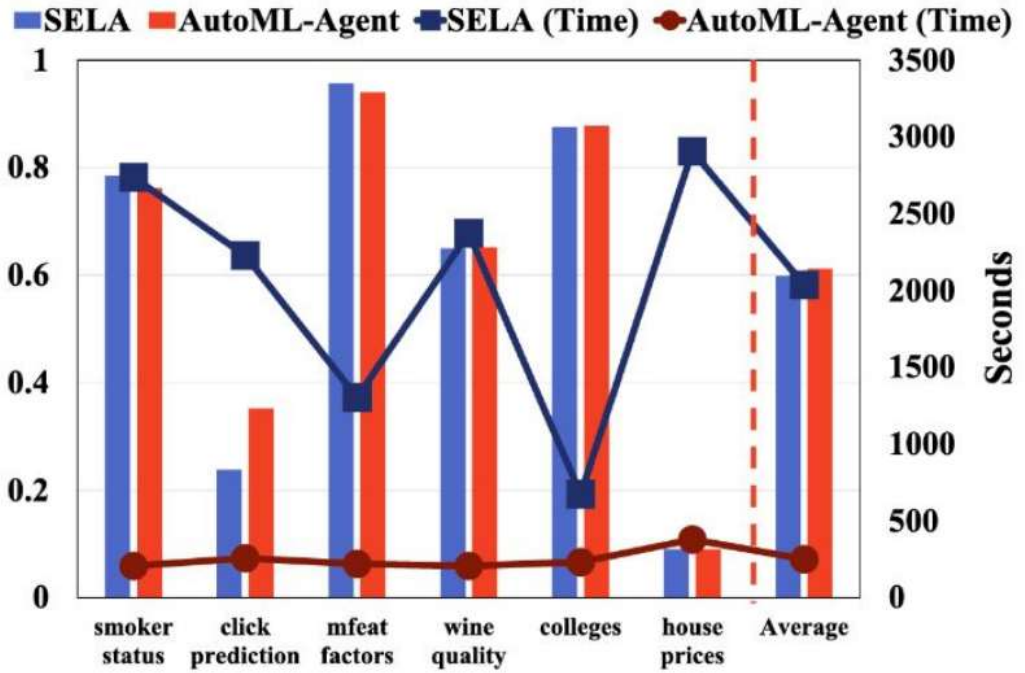


对七个下游任务使用十四个数据集的评估表明，AutoML-Agent 成功地编写了可执行代码，并且生成的模型性能超过了人类构建的模型。

结果评估

消融研究（左侧）表明，计划分解和验证阶段确实有效，超越了基于简单检索的规划的性能。





与基于训练的方法如 SELA（右侧）相比，AutoML-agent框架速度快 8 倍，同时在减少计算资源和时间的情况下实现了类似或更好的结果。

内容

- 背景
- 系统设计
- 评估实验
- 思考

思考

不足之处：

1.实验环境依赖高性能硬件（如 Ubuntu 22.04 LTS 服务器，配备 8 个 NVIDIA A100 GPU 和 Intel Xeon Platinum 8275CL CPU），降低了实用性。

2.实验规模有限仅涵盖七个下游任务（尽管涉及五种模态），可能无法代表更广泛的应用场景。

可取之处：

1.指令解析，使用户查询更容易被LLM理解，从而生成优质的全局计划。

2.多阶段验证，使得用户查询更加明晰，执行流程更加符合用户要求。

3.检索增强，调用API，使生成计划所用的知识源不再局限于LLM的固有知识。

4.计划分解，避免冗余的全局计划影响子环节，子任务的引入使每个环节的行动更加明确。

Q&A

Presenter: Jinhan Xin

2025.09.18