ADP-MA: An Interactive System for Autonomous Data Processing using Meta-Agents

Udayan Khurana*
IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
ukhurana@us.ibm.com

Abstract

We demonstrate **ADP-MA** (Autonomous Data Processing using Meta-Agents), a system that autonomously solves a complex and diverse set of data processing tasks. Three domain-agnostic *meta-agents* coordinate task-specific *ground agents* through a multi-stage pipeline: data understanding, planning, critique, expansion, execution, and finalization. Errors are caught early via progressive sampling on small data subsets before running on full data. The system supports three execution strategies, twelve domain knowledge packs, and confidence-based early stopping. An interactive web interface lets users watch pipelines being built in real time, replay completed runs at any stage, and compare results across cases. On four benchmarks, ADP-MA reaches 90.6% on DSEval, 44.8% on KramaBench, 50.0% on DA-Code, and 70.0% on AgentBench, outperforming published single-agent baselines.

1 Introduction

Building data processing pipelines requires domain knowledge and a variety of programming and analytical skills. The process is time-consuming from development to maintenance. LLM-based coding assistants often fail to adapt to specialized domain needs. While they code effectively, they cannot yet plan multi-stage workflows, detect silent data corruption, or recover when a plan turns out to be wrong.

Several agentic systems have tackled parts of this problem. AutoKaggle [Li *et al.*, 2024b] chains five agents through a fixed phase sequence. smolagents [Roucher and others, 2025] puts a frontier LLM in a flat execution loop but offers no plan-level recovery or monitoring. DS-GURU [Lai *et al.*, 2025] and similar single-agent approaches [Chen and others, 2025] skip hierarchical orchestration entirely. DS-Agent [Guo *et al.*, 2024] relies on case-based retrieval from prior solutions rather than runtime planning and monitoring, while TaskWeaver [Qiao *et al.*, 2023] provides a code-first agent framework but leaves multi-stage decomposition and recovery to the user. General multi-agent frameworks such as AutoGen [Wu *et al.*, 2023] and MetaGPT [Hong *et al.*, 2024] offer conversational and role-based orchestration but leave data-specific concerns (schema handling, progressive sampling, domain knowledge) to the application. AutoML-Agent [Trirat *et al.*, 2025] targets full-pipeline AutoML through retrieval-augmented planning and parallel decomposition, but specialises in model search and training rather than the broader data-processing tasks ADP-MA addresses. AutoDCWorkflow [Li *et al.*, 2024a] targets only one pipeline stage. SPIRAL [D'Alessandro *et al.*, 2024] introduces a critic mechanism that we adapt for early stopping. Notebook copilots (GitHub Copilot, Amazon CodeWhisperer) assist at the snippet level, while visual builders (KNIME, RapidMiner) require manual assembly.

ADP-MA[1] takes a different approach: it separates *orchestration* from *execution*. Three persistent meta-agents (Orchestrator, Architect, Monitor) handle planning, critique, and oversight without encoding any domain logic. For each pipeline stage, they create short-lived ground agents with the right tools, schema contracts, and domain knowledge packs. The meta-agents stay the same across problems; the ground agents are disposable and specialized. Progressive sampling, two-level backtracking, and rule-based monitoring catch problems early. Because every stage is visible in the interface, users can inspect plans, intermediate outputs, and generated code instead of receiving an opaque final answer.

This paper presents the ADP-MA demo interface, which lets users observe meta-agent decisions in real time, trace from plans to generated code, replay any stage of a completed run, and compare results across cases. The underlying architecture is described in [Khurana, 2026a]. Across four benchmarks, these architectural mechanisms consistently improve results regardless of which LLM is plugged in.

2 System Architecture

Figure 1 shows the architecture. The design rests on four ideas: decompose a task into a plan before generating any code; critique that plan before executing it; test generated code on small data samples first; and keep all domain knowledge out of the orchestration layer so that the same meta-agents work across problems.

*ORCID: https://orcid.org/0000-0001-8113-1210

[1]Video demonstration: https://youtu.be/dVA2WW1JOY8r.

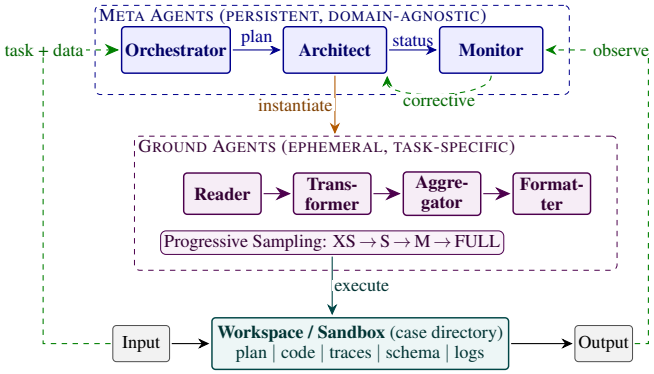

Figure 1: ADP-MA architecture. Three meta-agents coordinate ground-level agents through a six-stage workflow with process-level sandboxing. Progressive sampling ensures early error detection.

Meta-Agents. Three meta-agents persist across tasks. The **Orchestrator** profiles the input data, decomposes the task, and maintains shared context. The **Architect** maps plan phases to agent assignments and handles backtracking when something fails. The **Monitor** checks for problems like row-count explosions and null-rate spikes using deterministic rules, without making any LLM calls.

Ground-Level Agents. Ground agents are short-lived and task-specific. Each one is created on demand with a type (DataProcessor, Aggregator, or AnswerFormatter), a set of tools, and a schema contract specifying expected inputs and outputs. It runs in a sandboxed process, generates Python code, and returns structured results. Adding support for a new domain means writing a knowledge pack, not modifying the meta-agents.

Pipeline Stages. A task typically moves through: (1) Data Understanding (schema inference and statistics; see [Khurana, 2026b] for the schema-grounded variant used here), (2) Planning (decomposition into phases), (3) Critique (severity-gated validation, up to 2 rounds), (4) Expansion (phases broken into substeps with agent assignments), (5) Execution (code generation with progressive sampling, and (6) Finalization (pipeline assembly and documentation).

Other Mechanisms. Three execution strategies let users trade reliability for speed: **centralized** runs agents serially with full context, **autonomous** runs them in parallel with only contract-based context sharing, and **hybrid** groups agents by phase. Twelve domain knowledge packs inject guidance when keywords in the data or task match a known domain. Schema contracts enforce column-level type checks between agents. Two-level backtracking retries code locally or rewrites the plan globally. ADP-MA works with any LLM, and users can pick separate models for planning and code generation. The system is built on the Google Agent Development Kit [Google, 2025]. Each of the twelve domain packs is 100–400 lines of Python (keywords for activation, code skeletons for recurring operations, optional output validators) and takes a few hours to author with no retraining required.

3 Demonstration Interface

The front end is built with React and communicates with a FastAPI back end over WebSockets, so events stream to the browser as they happen. There are three main views.

User Experience Flow. A user uploads CSV files, describes the task in natural language, and optionally picks different LLMs for planning and code generation (e.g., DeepSeek-3.2 for planning, Sonnet 4.5 for coding) along with an execution strategy. Clicking "Run" starts the pipeline. A progress bar tracks the stages, and cards within each stage show what individual agents are doing. After the run, users can download the output, browse the assembled pipeline, or switch to Replay Mode to step through what happened.

Live Mode. As the pipeline runs, stage cards update with color-coded status (green, yellow, red). Users can click any card to expand it and inspect the input schema, generated code, or execution logs for that agent. Small dots next to each agent indicate which sample level it has reached. Domain pack badges show which packs were auto-activated, and monitor alerts appear inline when something looks off (e.g., an unexpected row-count increase). A code viewer displays the generated Python with syntax highlighting and diffs between revisions. Users can also pause an ongoing run to examine intermediate outputs before the pipeline continues.

Replay Mode. Any completed run can be replayed at speeds from 0.5× to 10×. Clicking a stage or agent jumps directly to that point. The progress bar doubles as a seekable timeline with timing annotations, and the layout is identical to Live Mode, making it easy to review or present past runs without re-executing them.

Analytics Dashboard. A separate dashboard aggregates metrics across all runs: total cases, success rates, and durations. It includes daily trend charts, an LLM comparison table (ranked by pass rate/cost), execution strategy breakdowns, failure categories with recovery rates, and stage-timing waterfalls that separate LLM time from other processing.

4 Evaluation Highlights

Table 1 reports results on four benchmarks with up to five LLMs. On DSEval [Zhang *et al.*, 2024], ADP-MA reaches 90.6% (+14.4pp over CoML+GPT-4). On KramaBench [Lai *et al.*, 2025], 44.8% with Sonnet 4.5 surpasses smolagents [Roucher and others, 2025]+GPT-o3 (41.4%) and more than doubles DS-GURU+GPT-o3 (22.1%); at a fixed LLM, ADP-MA + GPT-5 reaches 42.9% (still ahead of both GPT-o3 baselines), and on AgentBench ADP-MA + GPT-4o reaches 56.0% versus 32.0% for GPT-4 alone—the architecture, not the model, accounts for the gap. On AgentBench [Liu *et al.*, 2024], both GPT-5 and Sonnet 4.5 reach 70.0%. DA-Code [Huang *et al.*, 2024] shows similar margins over standalone GPT-4. For context, human experts on KramaBench average 71%, with 46% of their failures attributed to incorrect pipeline design [Lai *et al.*, 2025]—the stage where ADP-MA's hierarchical planning helps most. The gap to the human baseline concentrates by domain: with Sonnet 4.5, ADP-MA reaches **67%** on biomedical (6/9), **61%** on legal (19/31), and **52%** on wildfire (11/21) where the activated

		ADP-MA +				
Benchmark	**N**	**DS-3.2**	**GPT-4o**	**GPT-5**	**Sonnet**	**Gemini**
DSEval	299	89.6%	81.3%	83.6%	**90.6%**	86.6%
KramaBench	105	41.0%	26.7%	42.9%	**44.8%**	38.5%
DA-Code	52	48.1%	32.7%	42.3%	44.2%	**50.0%**
AgentBench	100	61.0%	56.0%	**70.0%**	**70.0%**	65.0%

Table 1: Cross-benchmark results (DS-3.2 = DeepSeek-3.2, Sonnet = Claude Sonnet 4.5, Gemini = Gemini 2.5 Pro). Best per benchmark in bold.

packs cover the needed operations, but only **35%** on environment (7/20), **25%** on archaeology (3/12), and **8%** on astronomy (1/12)—domains where physics-derived computations, spatial-coordinate transformations, and specialized scientific operations fall outside the current twelve packs. This points to pack coverage, not orchestration, as the principal remaining gap. The ranking across models does not follow the usual capability ordering, indicating that schema contract compliance matters as much as the choice of LLM.

Variance. Run-to-run stability (AgentBench, 3 repeats, $T{=}0$): GPT-4o 52.3% $\pm$3.2pp, DeepSeek-3.2 61.3% $\pm$2.3pp, Sonnet 4.5 69.3% $\pm$1.2pp; 93% of Sonnet 4.5 tasks produced identical outcomes across runs vs. 78–79% for the other two. Full breakdowns are in [Khurana, 2026a].

5 Demonstration Scenarios

We walk through five cases drawn from real application domains—environmental monitoring, regulatory compliance, digital humanities, public health, and astronomy—that exercise different parts of the architecture, ranging from 31 seconds to 40 minutes.

Wildfire burned area (2 agents, 53s). A user uploads wildfire data and asks for average burned area by cause and month. The wildfire domain pack activates automatically, the Architect produces a two-phase plan, and critique flags missing cause values as MINOR, triggering one refinement. Execution progresses through sample levels to full data.

FTC fraud count (2 agents, 275s). A legal task requiring a join across a primary DataFrame and three supplementary sheets. The legal domain pack activates, and schema contracts route data correctly through the multi-source join.

Roman city proximity (3 agents, 775s). Spatial joins across two archaeological datasets fail repeatedly at the code level. After local retries exhaust their budget, the Monitor escalates critique severity from MINOR to MAJOR, signaling that the problem is in the plan, not the code. The Architect discards the original approach and generates a new plan with different agent assignments—illustrating two-level backtracking between fixable code errors and flawed plans.

Beach water quality (2 agents, 404s). A temporal pivot fails on the smallest sample. The system goes through 11 code revisions, each caught on a 1% sample, and still finishes in under 7 minutes.

Stellar populations (6 agents, 2418s). A KramaBench astronomy task requiring multi-step, physics-derived computations across sub-regions. The astronomy pack activates, but the initial parallel plan produces inconsistent unit conventions across sub-agents; the Monitor escalates critique from MAJOR to CRITICAL, signalling a coordination-level fault, not a local code error. The Architect rebuilds it as a six-stage sequential pipeline with an explicit unit-contract between agents, completing at full data in $\approx$40 minutes—the system's deepest critique-escalation path (MINOR $\rightarrow$ MAJOR $\rightarrow$ CRITICAL). A single-agent DSEval task (31 s) is also preloaded for minimal-overhead simple problems.

Demo Setup and Deployment. The system runs on a laptop connected to cloud LLM APIs (DeepSeek, Anthropic, Google, OpenAI). Simple tasks complete in 30–60 seconds; complex multi-agent cases can take several minutes. The live portion uses the 53-second wildfire case; attendees may upload their own CSVs, and a Replay Mode fallback plays preloaded cases without API calls. Each run is a self-contained *case directory* on disk holding the plan, per-substep code revisions, execution traces, monitor alerts with timestamps, and profiling; every emitted event carries a unique ID and stage tag, so logs are independently verifiable and reconstructible by a third party.

Who Should Attend? Practitioners who spend time on repetitive data cleaning, reshaping, or joining will find ADP-MA productive; researchers studying multi-agent coordination can watch the Architect plan, the Monitor flag problems, and backtracking reshape both code and plans.

Limitations. ADP-MA depends on cloud LLM availability and provides no formal correctness guarantees on generated code; outputs are drafts to inspect. Three recurring failure observed are: (i) tasks requiring specialized physics or geospatial reasoning beyond the current twelve packs; (ii) plans that pass critique but produce semantically wrong outputs that shape-based monitors do not catch; and (iii) refinement loops that exhaust their budget on the same error class.

6 Conclusion

The results across four benchmarks show that separating persistent meta-agents from disposable ground agents pays off, and that the benefit holds across different LLMs rather than depending on a particular model. The demo interface makes this separation tangible: users can watch plans form, see critique reshape those plans, and inspect generated code. A natural next step is adaptive routing, where the meta-agent layer picks different models per pipeline stage.

Use of Generative AI: We used Generative AI tools to polish this write-up and fix several code-related issues in our system. All the ideas, system design, general implementation, and initial write-up of the paper were performed by the author.

References

[Chen and others, 2025] Zui Chen et al. A survey on data science with large language models. *arXiv preprint arXiv:2505.03418*, 2025.

[D'Alessandro *et al.*, 2024] Francesco D'Alessandro, Zhou He, and I-Hung Huang. SPIRAL: Improving code generation through self-verification and progressive refinement in multi-agent LLM systems. *arXiv preprint arXiv:2512.23167*, 2024.

[Google, 2025] Google. Google agent development kit (ADK). https://google.github.io/adk-docs/, 2025. Accessed: 2026.

[Guo *et al.*, 2024] Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. DS-Agent: Automated data science by empowering large language models with case-based reasoning. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[Hong *et al.*, 2024] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[Huang *et al.*, 2024] Yue Huang, Qihui Luo, Jiawen Li, et al. DA-Code: Agent data science code generation benchmark. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024.

[Khurana, 2026a] Udayan Khurana. Autonomous data processing using meta-agents. *arXiv preprint arXiv:2602.00307*, 2026.

[Khurana, 2026b] Udayan Khurana. Data understanding for agents using schema grounding. DEEM '26, New York, NY, USA, 2026. Association for Computing Machinery.

[Lai *et al.*, 2025] Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Om Chabra, Sivaprasad Sudhir, Anna Zeng, Anton A. Zabreyko, Chenning Li, Ferdi Kossmann, Jialin Ding, Jun Chen, Markos Markakis, Matthew Russo, Weiyang Wang, Ziniu Wu, Michael J. Cafarella, Lei Cao, Samuel Madden, and Tim Kraska. KRAMABENCH: A benchmark for knowledge-intensive, real-world, multi-step data science pipelines. *arXiv preprint arXiv:2506.06541*, 2025.

[Li *et al.*, 2024a] Lan Li, Liri Fang, Bertram Ludäscher, and Vetle I. Torvik. AutoDCWorkflow: LLM-based data cleaning workflow auto-generation and benchmark. *arXiv preprint arXiv:2412.06724*, 2024.

[Li *et al.*, 2024b] Ziming Li, Qianbo Zang, David Ma, Jiawei Guo, Tuney Zheng, Minghao Liu, Xinyao Niu, Yue Wang, Jian Yang, Jiaheng Liu, Wanjun Zhong, Wangchunshu Zhou, Wenhao Huang, and Ge Zhang. AutoKaggle: A multi-agent framework for autonomous data science competitions. *arXiv preprint arXiv:2410.20424*, 2024.

[Liu *et al.*, 2024] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Zhang, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[Qiao *et al.*, 2023] Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, et al. TaskWeaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*, 2023.

[Roucher and others, 2025] Aymeric Roucher et al. smolagents: a smol library to build great agents. https://huggingface.co/docs/smolagents, 2025. HuggingFace. Accessed: 2026.

[Trirat *et al.*, 2025] Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. AutoML-Agent: A multi-agent LLM framework for full-pipeline AutoML. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

[Wu *et al.*, 2023] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.

[Zhang *et al.*, 2024] Yuge Zhang, Qiyang Jiang, Xingyu Han, Nan Chen, Yuqing Yang, and Kan Ren. Benchmarking data science agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5636–5652, 2024.