PREFACE TO THE EDITION

The rapid evolution of data science and machine learning continues to redefine the way organizations, industries, and societies generate knowledge, solve complex problems, and make informed decisions. As artificial intelligence moves beyond experimental research into mission-critical applications, the demand for robust, interpretable, scalable, and trustworthy machine learning systems has never been greater. The current issue of the **Eduschool International Journal of Data Science and Machine Learning (EIJDSML)** brings together a carefully curated collection of scholarly contributions that examine these emerging directions, offering readers comprehensive insights into the latest theoretical advances and practical methodologies shaping the future of intelligent systems.

The opening article explores the integration of causal inference with modern machine learning, highlighting the crucial distinction between prediction and intervention. By examining frameworks such as potential outcomes, structural causal models, causal forests, and double machine learning, the study emphasizes how causal reasoning enables reliable decision-making across domains including healthcare, marketing, pricing, and public policy.

As machine learning systems increasingly operate in dynamic real-world environments, maintaining their performance has become an essential challenge. Addressing this concern, the second contribution investigates Data-Centric AI and MLOps, focusing on continuous monitoring, data validation, drift detection, and model governance. The article demonstrates how modern MLOps practices improve the reliability, reproducibility, and long-term sustainability of deployed machine learning systems.

Forecasting has also undergone a remarkable transformation with the emergence of foundation models. The third article reviews Time-Series Foundation Models for Zero-Shot Forecasting, presenting recent developments that enable pretrained models to generate accurate forecasts across previously unseen datasets without task-specific retraining. Through an examination of state-of-the-art architectures and comparative evaluations, the paper illustrates how foundation models are reshaping predictive analytics while identifying opportunities for future research.

Financial security remains one of the most important applications of machine learning, and the fourth article provides an extensive review of Graph Neural Networks for Financial Fraud and Anomaly Detection. By leveraging relational structures within transaction networks, graph-based learning models significantly improve the identification of coordinated fraudulent activities. The discussion of heterogeneous graphs, temporal networks, imbalance-aware architectures, and explainable graph learning demonstrates the growing importance of graph intelligence in financial analytics.

The concluding article examines one of the most transformative developments in artificial intelligence—LLM-powered agentic data science. Moving beyond conventional language models, the paper investigates autonomous AI agents capable of planning analyses, generating code, executing tools, validating outputs, and refining results through iterative reasoning. Alongside these remarkable capabilities, the study thoughtfully discusses challenges including reproducibility, hallucination, verification, and safe deployment, reinforcing the importance of responsible AI practices in automated scientific analysis.

Collectively, the articles in this issue reflect the ongoing transition from predictive algorithms toward intelligent, adaptive, and trustworthy data-driven systems. They demonstrate that future advances in machine learning will depend not only on increasing computational capability but also on improving interpretability, causal understanding, operational reliability, collaborative intelligence, and responsible deployment. Together, these contributions provide valuable perspectives for academics, researchers, practitioners, and students seeking to understand the evolving landscape of modern data science.

We extend our sincere appreciation to all authors for their valuable scholarly contributions and to the reviewers whose expertise and constructive evaluations have ensured the high academic standards of this publication. We also thank our Editorial Board members for their continued guidance and commitment to excellence. We hope that this issue of Eduschool International Journal of Data Science and Machine Learning (EIJDSML) serves as a meaningful resource that stimulates further research, interdisciplinary collaboration, and innovative applications in the rapidly advancing fields of data science, machine learning, and artificial intelligence.

Dr. Ginne M James

Chief Editor

CONTENTS

SL. NO.	TITLE	AUTHOR	PAGE NO.
1	Causal Inference and Machine Learning for Reliable Decision-Making	Rasmi M	1-7
2	Data-Centric AI and MLOPS: Drift Detection and Model Monitoring	V Sakhtipriyanka	8-14
3	Time-Series Foundation Models for Zero-Shot Forecasting	Win Mathew John	15-20
4	Graph Neural Networks for Financial Fraud and Anomaly Detection	Raji N	21-27
5	LLM-Powered Agentic Data Science: Automated Analysis and Insight Generation	M Keerthika	28-34

EDUSCHOOL INTERNATIONAL JOURNAL OF
DATA SCIENCE AND MACHINE LEARNING (EIJDSML)

(Open Access, Double-Blind Peer Reviewed Journal)

ISSN Online: 3139-7344

Causal Inference and Machine Learning for Reliable Decision-Making

Rasmi M

Assistant Professor, Department of Computer Applications, Bethania Institute of Management Studies, Kunnamkulam, Kerala, India

Article information

Received: 12th March 2026
Received in revised form: 15th April 2026
Accepted: 18th May 2026
Available online: 18th July 2026

Volume: 1
Issue: 2
DOI: https://doi.org/10.63090/EIJDSML/3139.7344.0006

Abstract

Machine learning systems trained to predict outcomes from observational data often fail when their outputs guide interventions. A model that accurately forecasts who will churn, default, or recover does not by itself reveal what happens if a policy changes the treatment assigned to those people. This gap between prediction and intervention is the central concern of causal inference. This paper reviews the methodological bridge connecting causal reasoning with modern supervised learning and explains why correlation-driven models give biased answers to questions about actions. The potential outcomes framework and structural causal models are introduced as complementary languages for stating the target of estimation. Identification is treated through directed acyclic graphs, the backdoor criterion, and instrumental variables, which specify when an effect can be recovered from data. The discussion then turns to estimation of average and conditional average treatment effects using meta-learners (S, T, and X), causal forests, and double or debiased machine learning, together with uplift modeling for targeting decisions. A synthetic benchmark illustrates typical error behavior: orthogonalized and forest-based estimators reach an absolute ATE bias near 0.05 and lower CATE error than a single-model baseline. Applications in pricing, marketing, healthcare, and public policy are surveyed, along with validation difficulties that arise because counterfactual outcomes are never observed. The paper closes with the assumptions, chiefly unconfoundedness and overlap, on which every reported number depends.

Keywords:- Causal Inference, Treatment Effect Estimation, Double Machine Learning, Causal Forests, Uplift Modeling, Confounding, Potential Outcomes.

.

I. INTRODUCTION

Supervised learning answers questions of the form: given features X, what value of the outcome Y should be expected? Such predictions support many useful tasks, from ranking search results to flagging fraudulent transactions. A different class of questions drives most operational decisions. A clinician asks whether a drug will reduce a patient's risk, not merely whether high-risk patients tend to receive it. A marketing team asks whether a discount causes a purchase, not whether discount recipients buy more. These are questions about interventions, and they cannot be settled by accuracy on held-out data [1], [2]. The difficulty is structural. Observational datasets record decisions that were already made, often by agents who acted on information correlated with the outcome. Patients in poor health receive aggressive treatment; price-sensitive shoppers receive coupons. A predictive model fitted to such data learns the joint behavior of the assignment process and the outcome process, and it cannot separate them without additional assumptions [3]. When the assignment mechanism shifts, as it does whenever a

new policy is deployed, the model's correlational structure no longer holds. Causal inference supplies the missing structure. It defines the estimand, the quantity a decision maker actually wants, and states the conditions under which that quantity can be recovered from data. Over the past decade a body of work has joined this classical theory with flexible machine learning, producing estimators that use neural networks, gradient boosting, and random forests as components while retaining valid statistical guarantees [4], [5], [6]. This paper reviews that synthesis. Section II presents the potential outcomes framework and structural causal models. Section III treats identification and confounding. Section IV reviews ML estimators for treatment effects. Section V reports a synthetic benchmark and discusses applications. Section VI examines assumptions and limitations, and Section VII concludes.

II. FOUNDATIONS: POTENTIAL OUTCOMES AND STRUCTURAL CAUSAL MODELS

A. Potential Outcomes

The Neyman-Rubin potential outcomes model assigns to each unit i two outcomes: $Y_i(1)$, the outcome under treatment, and $Y_i(0)$, the outcome under control [1], [7]. The individual treatment effect is the difference $Y_i(1) - Y_i(0)$. Only one of the two potential outcomes is ever observed, since a unit is either treated or not, a feature Holland termed the fundamental problem of causal inference [8]. The observed outcome relates to the potential outcomes through the assignment indicator:

$$Y_i = T_i Y_i(1) + (1 - T_i) Y_i(0) \tag{1}$$

Because individual effects are unidentifiable without strong assumptions, attention centers on averages. The average treatment effect (ATE) is the expectation of Y(1) - Y(0) over the population. The conditional average treatment effect (CATE), written:

$$\tau(x) = E[Y(1) - Y(0) \mid X = x] \tag{2}$$

This describes how the effect varies with covariates and is the target of personalized decision rules [4], [9]. A valid interpretation of any estimate rests on the stable unit treatment value assumption, which rules out interference between units and multiple versions of treatment [7].

B. Structural Causal Models and DAGs

Pearl's structural causal model represents a system as a set of variables and functions, each variable computed from its direct causes and an independent noise term [2], [10]. The qualitative skeleton of such a model is a directed acyclic graph (DAG), in which an arrow from A to B states that A appears in the function generating B. The do-operator formalizes intervention: do(T = t) replaces the function for T with the constant t, severing the arrows into T while leaving the rest of the system intact. The interventional distribution P(Y | do(t)) is generally different from the conditional distribution P(Y | t), and the difference is exactly what confounding measures. Fig. 1 shows a compact model with a confounder X that influences both treatment and outcome, a mediator M on the path from treatment to outcome, and an unobserved factor U. The graph makes the estimand and its obstacles visible at a glance: the total effect of T on Y travels along the direct edge and through M, while the backdoor path through X, and any path through U, must be blocked for the effect to be identified.

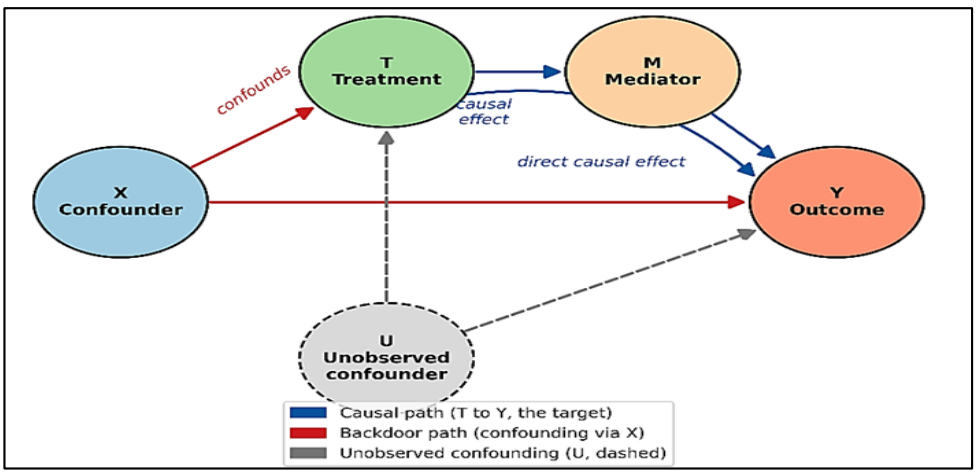

Fig. 1: A structural causal graph with treatment T, confounder X, mediator M, outcome Y, and unobserved factor U. Blue marks the causal path the analyst wants; red marks the backdoor path through X that must be blocked; dashed grey marks unobserved confounding. Takeaway: the graph names exactly what to estimate and what to adjust away.

III. IDENTIFICATION AND CONFOUNDING

Identification asks whether a causal quantity can be written as a function of the observed distribution. It is a property of the assumed model, not of the sample size, and no amount of data repairs a non-identified estimand. The clearest illustration is confounding. When a common cause influences both treatment and outcome, the naive difference in observed means mixes the treatment effect with the confounder's imprint.

Fig. 2 demonstrates the reversal that confounding can produce. In the simulated cohort, disease severity drives both the decision to treat and the outcome. Comparing treated and untreated patients directly suggests that treatment worsens the outcome, because sicker patients are treated more often. Stratifying on severity reveals the true effect, which is protective. This sign reversal is a finite-sample analogue of Simpson's paradox and a reminder that adjustment, not raw correlation, carries the causal content [11].

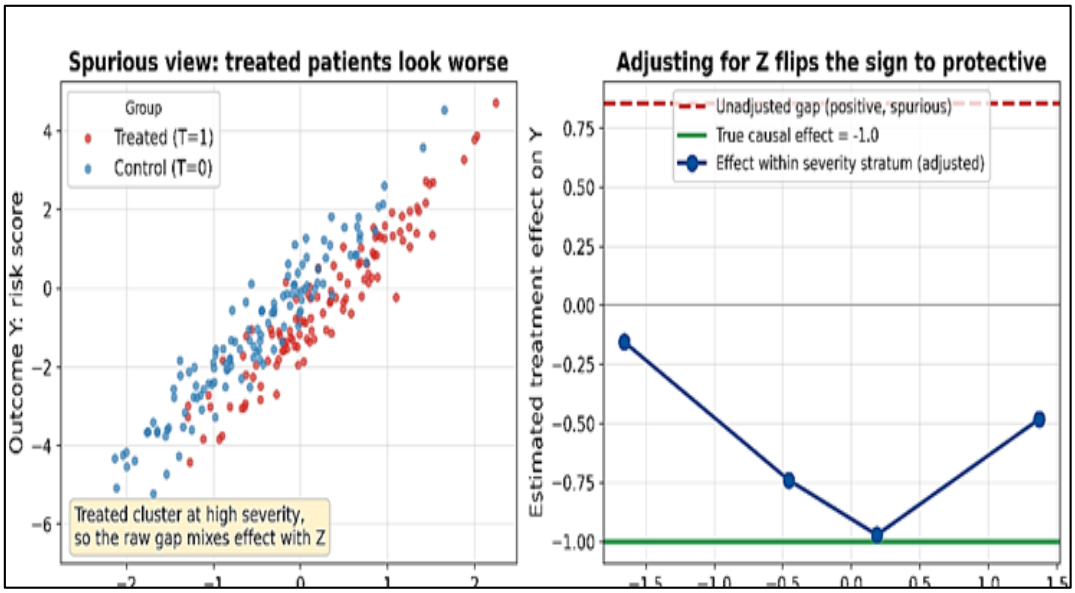

Fig. 2: A confounder reverses an apparent association. The unadjusted comparison (left) suggests harm because treated patients are sicker; stratifying on severity (right) recovers the protective effect. Takeaway: the sign of an effect can flip once the confounder is held fixed.

A. The Backdoor Criterion

A set of covariates Z satisfies the backdoor criterion relative to (T, Y) if no node in Z is a descendant of T and Z blocks every path from T to Y that begins with an arrow into T [2]. When such a Z is observed, the interventional distribution is identified by adjustment: P(Y | do(t)) equals the average over Z of P(Y | t, Z). Conditioning on the right set removes confounding, while conditioning on a mediator or a collider can introduce bias, so the choice of adjustment set follows from the graph rather than from a search for predictive features [12].

B. Instrumental Variables

When an unobserved confounder remains, as with U in Fig. 1, backdoor adjustment is unavailable. An instrumental variable offers an alternative route [13]. An instrument affects treatment, is independent of the unobserved confounders, and influences the outcome only through treatment. Variation in the instrument induces variation in treatment that is free of confounding, and under a monotonicity condition it identifies a local average treatment effect for the subpopulation whose treatment responds to the instrument [14]. Encouragement designs, policy discontinuities, and randomized eligibility are common sources of instruments in applied work.

IV. MACHINE LEARNING ESTIMATORS FOR TREATMENT EFFECTS

Once an estimand is identified, the remaining task is estimation from finite data. Classical methods such as matching and inverse propensity weighting remain useful, but they struggle with many covariates and complex response surfaces. Machine learning supplies flexible function approximators, and recent estimators wrap those approximators in procedures that preserve statistical validity [4], [5], [6].

A. Meta-Learners

Meta-learners reduce CATE estimation to standard regression [9]. The S-learner trains one model on covariates and treatment jointly, then differences its predictions at T = 1 and T = 0. It is simple but tends to shrink the estimated effect toward zero when the base learner regularizes the treatment indicator. The T-learner fits separate models for treated and control groups, which avoids that bias but wastes data when one group is small. The X-learner adds a cross step that imputes individual effects and reweights by the propensity score, improving accuracy under imbalanced assignment, a common situation in observational studies.

B. Causal Forests

Generalized random forests estimate heterogeneous effects by growing trees that split to maximize differences in treatment effect rather than differences in outcome [4], [15]. Honest sample splitting, in which one subsample chooses splits and another estimates effects within leaves, yields pointwise confidence intervals and asymptotic normality. Causal forests adapt to the covariates that drive heterogeneity and degrade gracefully when effects are uniform.

C. Double or Debiased Machine Learning

Double machine learning targets a low-dimensional effect while using arbitrary learners for the high-dimensional nuisance functions, namely the outcome regression and the propensity score [5]. Two devices protect the estimate. Neyman orthogonality makes the moment condition insensitive to small errors in the nuisance estimates, and cross-fitting removes the bias that arises from using the same data to fit nuisances and the effect. The result is a root-n consistent, asymptotically normal estimate of the ATE even when the nuisance learners converge slowly. Doubly robust scores combine the outcome model and the propensity model so that consistency holds if either is correct [16], [17].

D. Representation Learning and Deep Models

Neural approaches learn balanced representations in which treated and control distributions overlap. TARNet and counterfactual regression add an imbalance penalty between the two groups in representation space, reducing the error of individual effect estimates [18]. The causal effect variational autoencoder treats an unobserved confounder as a latent variable inferred from proxies [19]. These methods extend treatment effect estimation to images, text, and other unstructured covariates, at the cost of weaker theoretical guarantees than orthogonalized estimators provide.

Table 1. Machine learning estimators for treatment effects.

Estimator	Target	Base learner	Key property
S-learner	CATE	Any regressor	Simple; can bias effect toward zero
T-learner	CATE	Two regressors	Unbiased split; data-hungry
X-learner	CATE	Regressors + propensity	Strong under imbalance
Causal forest	CATE	Honest trees	Valid intervals; adapts to heterogeneity
Double ML	ATE	Any nuisance learner	Orthogonal; cross-fitted; root-n
TARNet / CFR	CATE	Neural network	Balanced representation; unstructured X

V. APPLICATIONS AND ILLUSTRATIVE RESULTS

A. A Synthetic Benchmark

To compare estimators on common ground, a synthetic dataset was generated with a known treatment effect, correlated covariates, confounded assignment, and heterogeneous response. Because the ground truth is fixed by construction, both ATE bias and the precision of conditional estimates can be measured directly. The figures reported here are illustrative of typical behavior and do not describe a deployed system. Fig. 3 summarizes absolute ATE bias and a CATE root mean squared error in the style of the PEHE metric across five estimators.

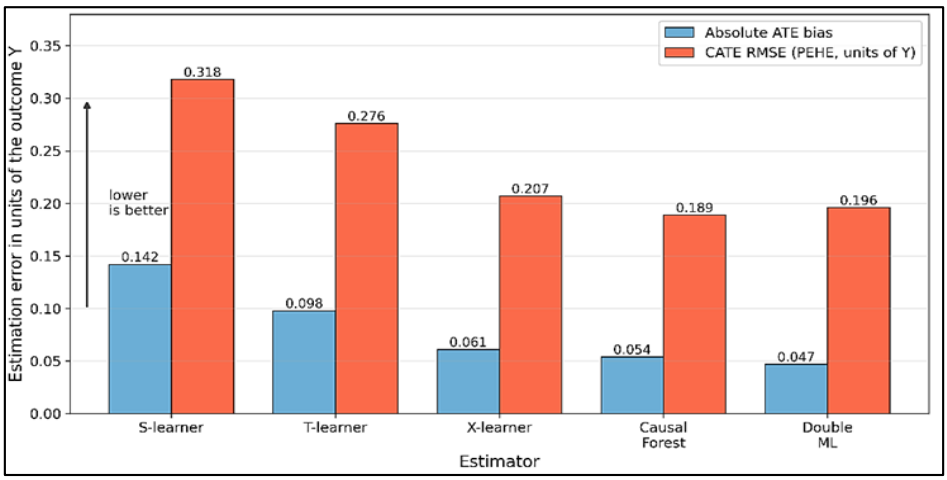

Fig. 3: Absolute ATE bias and CATE RMSE for five estimators on a synthetic benchmark with known ground truth. Lower bars are better. Takeaway: causal forests and double machine learning recover the true effect most accurately, while the S-learner is the most biased.

The pattern matches theory. The S-learner shows the largest bias because the base learner regularizes the treatment indicator. The X-learner improves markedly under the imbalanced assignment built into the simulation. Causal forests and double machine learning achieve the smallest ATE bias, near 0.05 in absolute terms, with the forest giving the lowest conditional error. Table 2 lists the numbers and adds an interval coverage column that reports how often nominal ninety-five percent intervals contain the truth.

Table 2. Synthetic benchmark results. Values are illustrative, not from a deployed study.

Estimator	|ATE bias|	CATE RMSE	95% interval coverage
S-learner	0.142	0.318	0.86
T-learner	0.098	0.276	0.89
X-learner	0.061	0.207	0.92
Causal forest	0.054	0.189	0.94
Double ML	0.047	0.196	0.95

B. Uplift Modeling for Targeting

Many business decisions need a ranking of who responds to an action, not a single average. Uplift modeling estimates the individual change in response probability and ranks the population by it [20], [21]. Targeting the high-uplift segment concentrates spending where treatment changes behavior, while sparing both the customers who would act anyway and those who react negatively. The Qini curve evaluates such rankings by plotting cumulative incremental response against the fraction of the population treated. Fig. 4 contrasts an uplift model and a causal forest with random targeting; the area between a model curve and the diagonal, the Qini coefficient, summarizes the gain.

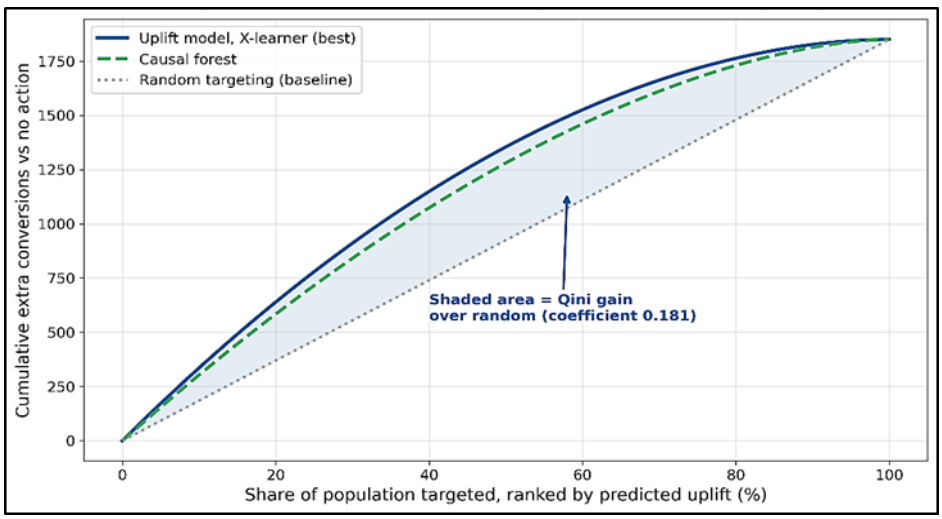

Fig. 4: Qini curve comparing uplift-ranked targeting with random targeting on a synthetic campaign. The shaded area above the diagonal is the Qini gain. Takeaway: ranking by predicted uplift earns more incremental conversions per unit of budget than treating people at random.

C. Domains

Causal estimators now appear across sectors. In pricing, demand response to price is a treatment effect, and confounding from inventory or seasonality biases naive elasticity estimates. In marketing, uplift models direct retention budgets toward persuadable customers. In healthcare, conditional effects guide who benefits from a therapy, with strong overlap requirements because clinicians do not assign treatment at random [22]. In public policy, program evaluation rests on quasi-experimental designs such as instruments and discontinuities, and machine learning estimators extend these to rich administrative data while preserving the credibility of the design [13], [23].

VI. ASSUMPTIONS AND LIMITATIONS

Every number in the previous section depends on assumptions that the data cannot verify. The first is unconfoundedness, also called ignorability, which states that treatment is independent of the potential outcomes given the observed covariates [7]. When an unmeasured factor influences both treatment and outcome, adjustment estimates remain biased, and the bias does not shrink with sample size. Sensitivity analysis quantifies how strong such a hidden factor would have to be to overturn a conclusion, and reporting it is now standard practice [24].

The second assumption is overlap, or positivity: every covariate profile must have a non-trivial probability of receiving each treatment. Where overlap fails, no method can compare like with like, and propensity weights explode. Diagnosing the overlap region and restricting inference to it is often wiser than extrapolating a model into covariate space with no support.

Validation poses its own difficulty. Because the counterfactual outcome is never observed, treatment effect estimates cannot be scored against a ground truth the way predictions can. Practitioners rely on synthetic benchmarks with known effects, on calibration of predicted effects within bins, on placebo and refutation tests that should return a null, and on agreement with randomized experiments where these exist [25], [26]. None of these substitutes for a held-out label, so causal evaluation remains an argument from converging evidence rather than a single accuracy figure.

VII. CONCLUSION

Reliable decisions need answers about interventions, and those answers do not follow from predictive accuracy alone. Causal inference provides the framework for stating what is being estimated and when it can be recovered, while machine learning supplies the flexible estimators that make the framework practical on high-dimensional data. Meta-learners, causal forests, double machine learning, and representation-based deep models now estimate average and conditional effects with controlled bias, as the synthetic benchmark illustrated. The remaining frontier is not algorithmic power but assumption management: stating unconfoundedness and overlap clearly, probing their fragility, and validating with the indirect evidence that causal questions allow. Teams that pair these estimators with honest assumption checks can move from models that describe the world to models that inform action.

REFERENCES

[1] D. B. Rubin, "Estimating causal effects of treatments in randomized and nonrandomized studies," J. Educ. Psychol., vol. 66, no. 5, pp. 688–701, 1974.

[2] J. Pearl, Causality: Models, Reasoning, and Inference, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 2009.

[3] G. W. Imbens and D. B. Rubin, Causal Inference for Statistics, Social, and Biomedical Sciences: An Introduction. New York, NY, USA: Cambridge Univ. Press, 2015.

[4] S. Wager and S. Athey, "Estimation and inference of heterogeneous treatment effects using random forests," J. Amer. Stat. Assoc., vol. 113, no. 523, pp. 1228–1242, 2018.

[5] V. Chernozhukov et al., "Double/debiased machine learning for treatment and structural parameters," Econometrics J., vol. 21, no. 1, pp. C1–C68, 2018.

[6] S. Athey and G. W. Imbens, "Machine learning methods that economists should know about," Annu. Rev. Econ., vol. 11, pp. 685–725, 2019.

[7] G. W. Imbens and D. B. Rubin, "Rubin causal model," in The New Palgrave Dictionary of Economics. London, U.K.: Palgrave Macmillan, 2008.

[8] P. W. Holland, "Statistics and causal inference," J. Amer. Stat. Assoc., vol. 81, no. 396, pp. 945–960, 1986.

[9] S. R. Kunzel, J. S. Sekhon, P. J. Bickel, and B. Yu, "Metalearners for estimating heterogeneous treatment effects using machine learning," Proc. Nat. Acad. Sci. U.S.A., vol. 116, no. 10, pp. 4156–4165, 2019.

[10] J. Pearl, "Causal diagrams for empirical research," Biometrika, vol. 82, no. 4, pp. 669–688, 1995.

[11] J. Pearl, "Comment: Understanding Simpson's paradox," Amer. Stat., vol. 68, no. 1, pp. 8–13, 2014.

[12] T. J. VanderWeele and I. Shpitser, "A new criterion for confounder selection," Biometrics, vol. 67, no. 4, pp. 1406–1413, 2011.

[13] J. D. Angrist, G. W. Imbens, and D. B. Rubin, "Identification of causal effects using instrumental variables," J. Amer. Stat. Assoc., vol. 91, no. 434, pp. 444–455, 1996.

[14] G. W. Imbens and J. D. Angrist, "Identification and estimation of local average treatment effects," Econometrica, vol. 62, no. 2, pp. 467–475, 1994.
[15] S. Athey, J. Tibshirani, and S. Wager, "Generalized random forests," Ann. Stat., vol. 47, no. 2, pp. 1148–1178, 2019.
[16] H. Bang and J. M. Robins, "Doubly robust estimation in missing data and causal inference models," Biometrics, vol. 61, no. 4, pp. 962–973, 2005.
[17] E. H. Kennedy, "Towards optimal doubly robust estimation of heterogeneous causal effects," Electron. J. Stat., vol. 17, no. 2, pp. 3008–3049, 2023.
[18] U. Shalit, F. D. Johansson, and D. Sontag, "Estimating individual treatment effect: Generalization bounds and algorithms," in Proc. 34th Int. Conf. Mach. Learn. (ICML), 2017, pp. 3076–3085.
[19] C. Louizos, U. Shalit, J. Mooij, D. Sontag, R. Zemel, and M. Welling, "Causal effect inference with deep latent-variable models," in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 30, 2017, pp. 6446–6456.
[20] N. J. Radcliffe and P. D. Surry, "Real-world uplift modelling with significance-based uplift trees," Stochastic Solutions White Paper, 2011.
[21] P. Gutierrez and J.-Y. Gerardy, "Causal inference and uplift modelling: A review of the literature," in Proc. Mach. Learn. Data Sci. Investment Manage., PMLR, vol. 67, pp. 1–13, 2017.
[22] N. Kallus, "Recursive partitioning for personalization using observational data," in Proc. 34th Int. Conf. Mach. Learn. (ICML), 2017, pp. 1789–1798
[23] S. Athey and G. W. Imbens, "The state of applied econometrics: Causality and policy evaluation," J. Econ. Perspect., vol. 31, no. 2, pp. 3–32, 2017.
[24] P. R. Rosenbaum and D. B. Rubin, "Assessing sensitivity to an unobserved binary covariate in an observational study with binary outcome," J. Roy. Stat. Soc., Ser. B, vol. 45, no. 2, pp. 212–218, 1983.
[25] A. Sharma and E. Kiciman, "DoWhy: An end-to-end library for causal inference," arXiv:2011.04216, 2020.
[26] C. Schuler, M. Baiocchi, R. Tibshirani, and N. Shah, "A comparison of methods for model selection when estimating individual treatment effects," arXiv:1804.05146, 2018.

EDUSCHOOL INTERNATIONAL JOURNAL OF
DATA SCIENCE AND MACHINE LEARNING (EIJDSML)

(Open Access, Double-Blind Peer Reviewed Journal)

ISSN Online: 3139-7344

Data-Centric AI and MLOPS: Drift Detection and Model Monitoring

V Sakhtipriyanka

Assistant Professor, Department of Computer Science, Kongunadu Arts and Science College, Coimbatore, India

Article information

Received: 23rd March 2026
Received in revised form: 27th April 2026
Accepted: 29th May 2026
Available online: 18th July 2026

Volume: 1
Issue: 2
DOI: https://doi.org/10.63090/EIJDSML/3139.7344.0007

Abstract

Machine learning systems in production fail in ways that training metrics rarely predict. Once a model serves live traffic, the data it receives drifts away from the distribution it learned, and accuracy degrades silently. This paper examines the shift from a model-centric view, where the dataset is fixed and architectures are tuned, toward a data-centric practice in which data quality, validation, and continuous monitoring become first-class engineering concerns. The discussion is organized around the machine learning operations (MLOps) lifecycle, covering continuous integration, delivery, and training, together with feature stores and model registries that make pipelines reproducible. A taxonomy of distribution shift is presented, separating covariate shift, prior or label shift, and concept drift, each with distinct detection needs. Detection methods are reviewed across two settings: batch statistics such as the population stability index, Kullback-Leibler and Jensen-Shannon divergence, the Kolmogorov-Smirnov test, and maximum mean discrepancy, and streaming detectors such as ADWIN and the drift detection method. Illustrative metrics show detection delay and false-alarm trade-offs across these methods, alongside a monitored stability index crossing an alert threshold that triggers retraining. Retraining strategies, governance, and current tooling are discussed. The aim is a practical reference for teams that must keep deployed models reliable as their input data changes.

Keywords:- Data-Centric AI, MLOps, Concept Drift, Covariate Shift, Model Monitoring, Population Stability Index, ADWIN, Retraining.

.

I. INTRODUCTION

A trained model that scores well on a held-out test set is only the beginning of a production system. The larger and more error-prone part of the system is the surrounding infrastructure for data collection, feature extraction, serving, and monitoring. Sculley and colleagues described this gap as hidden technical debt, noting that only a small fraction of a real-world machine learning system is the model code itself [1]. The remainder consists of configuration, data dependencies, and glue code whose failure modes are subtle and expensive.

Once deployed, a model operates under an assumption that is almost never true for long: that the data it receives at inference time follows the same distribution as the data it was trained on. Customer behavior changes, sensors age, upstream pipelines are modified, and adversaries adapt. The result is distribution shift, and its

practical consequence is a quiet decline in predictive quality that no offline metric will reveal until labels arrive, sometimes weeks later [2], [3].

Two responses have shaped recent practice. The first is data-centric AI, an emphasis on systematically improving the quality and consistency of data rather than only refining model architecture [4]. The second is machine learning operations, or MLOps, which adapts software engineering discipline to the lifecycle of models, adding continuous training to the familiar continuous integration and delivery [5], [6]. Drift detection and model monitoring sit at the intersection of these two ideas: they convert the abstract goal of reliable data into concrete signals that an operations team can act on.

This paper provides a structured account of that intersection. Section II reviews data-centric AI and the MLOps lifecycle. Section III sets out a taxonomy of drift. Section IV surveys detection and monitoring methods for both batch and streaming settings. Section V discusses retraining strategies, governance, and illustrative results. Section VI reviews tooling and open challenges, and Section VII concludes. Figure references and two summary tables support the discussion. Reported numbers are illustrative and are not drawn from a single deployed study.

II. DATA-CENTRIC AI AND THE MLOPS LIFECYCLE

For much of the past decade, progress was measured by holding a benchmark dataset constant and searching for better architectures. Data-centric AI inverts that emphasis. It treats the model as relatively fixed and asks how labeling consistency, coverage of rare cases, and removal of corrupted records change downstream accuracy [4]. In many applied settings, cleaning a noisy label set yields a larger gain than swapping one network for another, and the gain is easier to sustain.

MLOps gives this view an operational backbone. The lifecycle in Fig. 1 runs from data ingestion and validation, through a feature store and versioned datasets, to training, a model registry, and serving. What distinguishes it from a one-off training script is the closed loop: monitoring observes live data and predictions, drift detection raises alerts, and a retraining trigger feeds a continuous training pipeline that produces a new candidate model. Continuous integration tests code and data schemas, continuous delivery promotes validated models, and continuous training automates the refresh [5], [6].

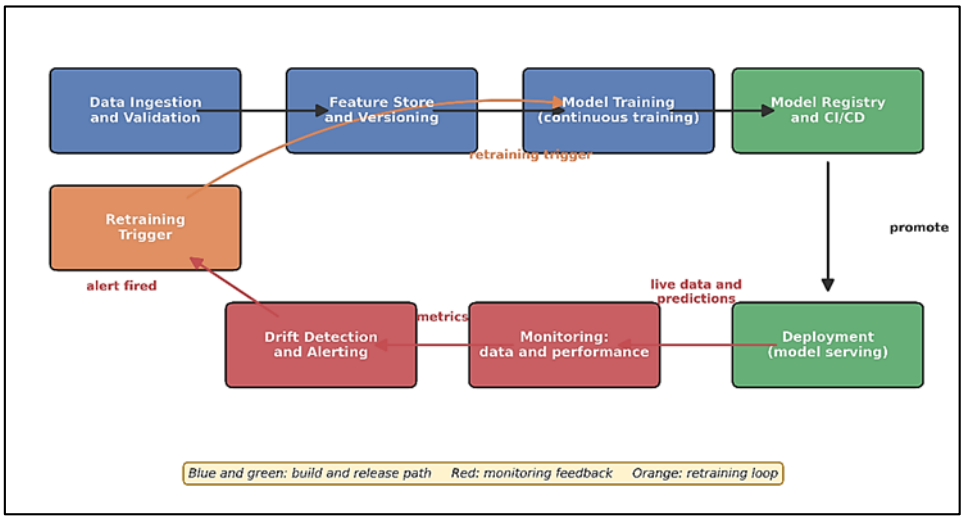

Fig. 1: MLOps lifecycle with a closed monitoring and retraining loop. Drift signals from production feed continuous training.

Several components make the loop reproducible. A feature store provides consistent feature computation for both training and serving, removing the skew that arises when offline and online code diverge [7]. A model registry records versions, lineage, and stage transitions, so that any served prediction can be traced to a specific model and dataset. Data validation tools encode expectations about schema, ranges, and distributions, and flag records that violate them before they reach the model [8], [9].

Quality of the engineering itself can be scored. Breck and colleagues proposed a rubric for production readiness, the ML Test Score, covering tests for features and data, model development, infrastructure, and monitoring [10]. Teams that adopt such rubrics tend to detect problems earlier, because the rubric forces explicit checks that are otherwise skipped under delivery pressure.

III. TYPES OF DRIFT

Distribution shift is not a single phenomenon. Let X denote inputs and y the target. Training assumes a joint distribution P(X, y); production may present a different joint distribution. Decomposing P(X, y) as P(y | X) P(X) or as P(X | y) P(y) clarifies which part has moved, and this distinction drives both detection and remedy [2], [11]. Fig. 2 organizes the categories.

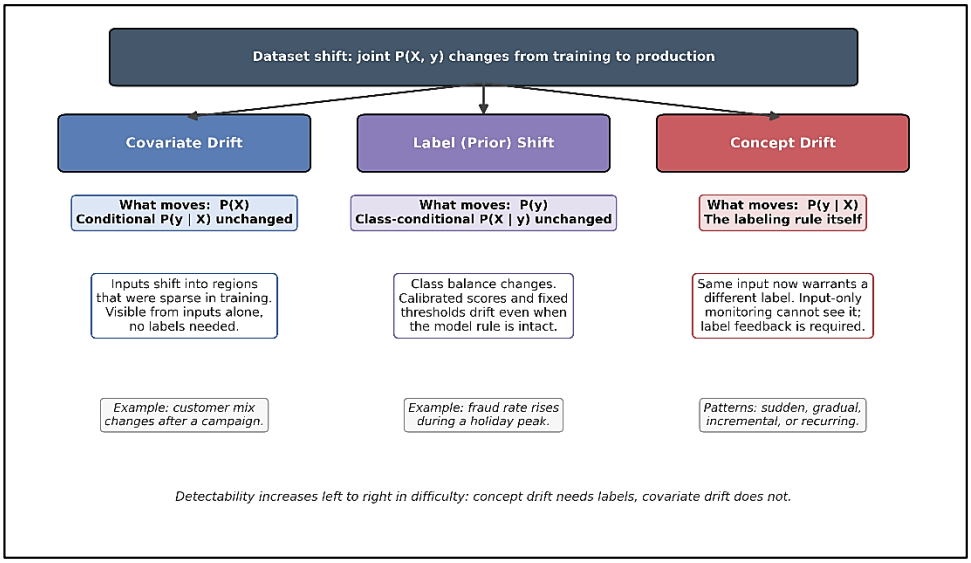

Fig. 2: Taxonomy of distribution shift, separating covariate shift, prior or label shift, and concept drift.

A. Covariate (Data) Drift

Covariate drift means P(X) changes while the conditional P(y | X) stays the same. The relationship the model learned is still valid, but it is queried in regions of input space that were sparse during training. A credit model trained before a change in customer demographics is a typical case. Covariate drift is the most directly observable form, because input features are available immediately, without waiting for labels [3], [12].

B. Prior (Label) Shift

Prior shift means the marginal P(y) changes while P(X | y) holds. The mix of classes seen in production differs from training, which distorts calibrated probabilities and any fixed decision threshold. Fraud rates that rise during a holiday season, or disease prevalence that changes across regions, produce prior shift. Detection often relies on estimating the label distribution from model outputs when true labels are delayed [11].

C. Concept Drift

Concept drift is the change of P(y | X) itself: the target relationship moves. The same input now warrants a different prediction. Gama and colleagues classify concept drift by its temporal pattern as sudden, gradual, incremental, or recurring [2]. Sudden drift follows a discrete event such as a policy change; gradual drift blends old and new concepts over time; recurring drift returns to a previous regime, as with seasonal patterns. Because concept drift alters the labeling rule, input-only monitoring cannot detect it, and label feedback or a proxy signal becomes necessary.

Table 1. Categories of distribution shift and their detection needs.

Drift type	What changes	Observable from inputs only	Typical remedy
Covariate (data)	P(X)	Yes	Reweighting, retraining
Prior (label) shift	P(y)	Partly (via outputs)	Threshold or prior correction
Concept drift	P(y | X)	No, needs labels	Retraining on recent data
Recurring concept	P(y | X), seasonal	No	Model ensembles, context flags

IV. DETECTION AND MONITORING METHODS

Detection methods divide into two families. Batch methods compare a current window of data against a fixed reference, usually a training or recent baseline sample. Streaming methods process examples one at a time and maintain an online estimate of change. The choice depends on latency budget, label availability, and the cost of a false alarm [13].

A. Batch Statistical Tests

The population stability index (PSI) bins a feature and sums the weighted log-ratio of bin proportions between reference and current samples. A common rule treats PSI below 0.10 as stable, 0.10 to 0.25 as moderate, and above 0.25 as a strong shift that warrants action. PSI is inexpensive and widely used in credit risk, though it is sensitive to bin choice. Kullback-Leibler divergence measures the information lost when the current distribution approximates the reference, and its symmetric, bounded variant, the Jensen-Shannon divergence, is preferred for monitoring because it stays finite when bins are empty [14].

For continuous features, the two-sample Kolmogorov-Smirnov test compares empirical cumulative distributions and returns a statistic and p-value, making it convenient for automated alerting. For high-dimensional or multivariate data, maximum mean discrepancy (MMD) compares distributions in a reproducing kernel Hilbert space and detects shifts that per-feature tests miss [15]. Rabanser and colleagues evaluated such tests for detecting shift and recommend combining a dimensionality reduction step with a statistical test, reporting that no single method dominates across shift types [13].

B. Streaming Detectors

When data arrives as a stream and rapid response matters, online detectors are used. The Drift Detection Method (DDM) tracks the online error rate of a classifier and signals a warning, then a drift, when the error rises by a set number of standard deviations above its minimum [16]. ADWIN maintains a variable-length window and automatically drops older sub-windows when their mean differs significantly from recent data, giving change detection with formal guarantees and no fixed window size [17]. These methods need a label or error signal, so they suit settings where feedback is fast.

C. Monitoring Data Quality and Performance

Drift detection is one layer of a broader monitoring stack. Data quality monitoring checks schema conformance, missing-value rates, and range violations at ingestion, catching pipeline faults that masquerade as drift [8], [9]. Performance monitoring tracks accuracy, precision, recall, calibration, and business metrics once labels are available, and uses proxy signals such as prediction confidence distributions while labels are pending. Fig. 3 shows a monitored stability index for a transaction-amount feature that stays low for several weeks, then climbs after an upstream change and crosses the alert threshold, triggering retraining.

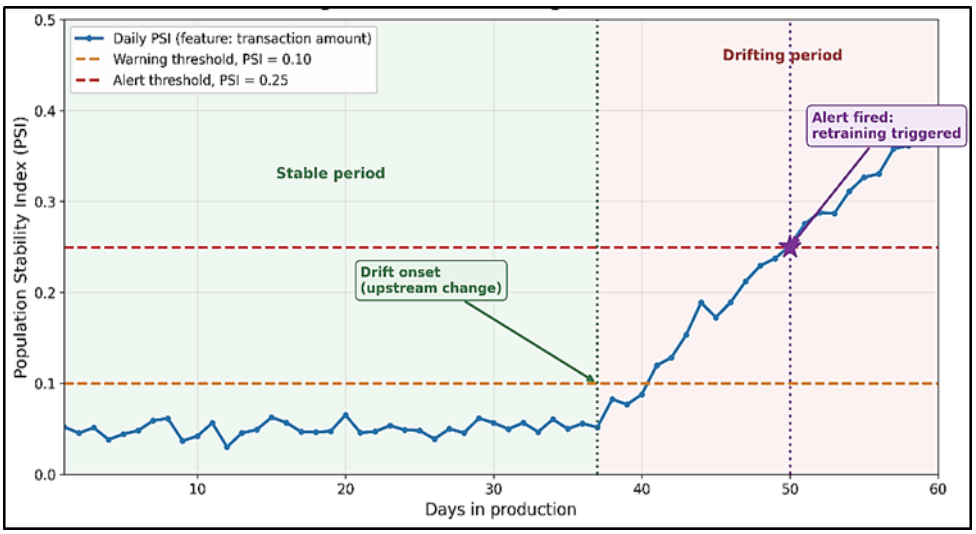

Fig. 3: Monitored population stability index for one feature crossing warning and alert thresholds, triggering a retraining event.

Method selection involves trade-offs between how quickly a true shift is detected and how often the monitor raises a false alarm. Fig. 4 gives illustrative values for mean detection delay and false-alarm rate across six methods. Streaming detectors such as ADWIN react with shorter delay than batch tests on the same stream,

while a poorly tuned detector can trade that speed for more spurious alerts. Table 2 summarizes the operating characteristics.

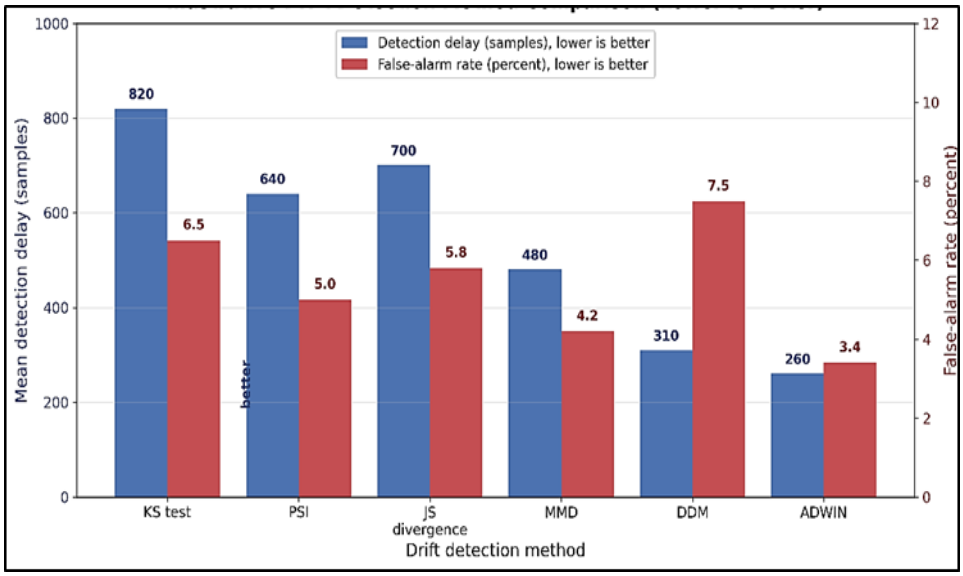

Fig. 4: Illustrative comparison of drift-detection methods by mean detection delay and false-alarm rate.

Table 2. Comparison of common drift-detection methods.

Method	Setting	Data type	Needs labels	Main use
PSI	Batch	Binned	No	Feature drift in credit risk
KL / JS divergence	Batch	Distributions	No	Distribution distance
KS test	Batch	Continuous	No	Per-feature shift test
MMD	Batch	Multivariate	No	High-dimensional shift
DDM	Streaming	Error stream	Yes	Online error monitoring
ADWIN	Streaming	Error / value stream	Yes	Adaptive windowed change

V. RETRAINING, GOVERNANCE, AND ILLUSTRATIVE RESULTS

An alert is useful only if a defined response follows. Retraining strategies fall along a spectrum. Scheduled retraining refreshes the model at fixed intervals regardless of drift, which is simple but wastes computation when data is stable and lags when it is not. Triggered retraining fires when a monitored metric crosses a threshold, as in Fig. 3, aligning cost with need. Online or incremental learning updates the model continuously from the stream, which suits fast-moving data but risks instability if noisy batches are absorbed without guardrails [2], [18].

A retraining decision should weigh the expected accuracy recovery against the cost and risk of deploying a new model. In the illustrative scenario of Fig. 3, accuracy had fallen by roughly nine points from a baseline of 0.91 by the time the alert fired, and triggered retraining on the most recent labeled window restored accuracy to within two points of the baseline. A new candidate is validated against a held-out recent set and a canary slice of live traffic before full promotion, so that a regression is caught before it reaches all users.

Governance turns these mechanics into a repeatable, auditable process. A model registry records which model is live, who approved it, and which dataset trained it. Rollback procedures let an operator revert to a prior version within minutes. For regulated domains, documentation artifacts such as model cards and datasheets record intended use, evaluation slices, and known limitations [19], [20]. Access controls and approval gates separate the act of training from the act of promotion, reducing the chance that an untested model reaches production.

Fairness and stability deserve continued monitoring after deployment, not only at launch. A model that was balanced across groups at training time can drift into disparate error rates as subpopulation distributions change, so slice-level metrics belong in the monitoring dashboard alongside aggregate accuracy [21]. Tying drift

alerts to these slices helps a team notice when degradation concentrates in a sensitive group rather than spreading evenly.

VI. TOOLING AND OPEN CHALLENGES

A practical ecosystem has formed around these ideas. Open-source libraries provide drift tests and monitoring reports, including Evidently for data and prediction drift dashboards [22] and the Alibi Detect library for outlier and drift detection with MMD and related tests [23]. Data validation frameworks such as TensorFlow Data Validation and Great Expectations encode schema and distribution checks in the pipeline [8], [24]. Experiment tracking and registry tools record runs, parameters, and model lineage. Streaming detectors are available through libraries for online learning, where ADWIN and DDM are standard components.

Open problems remain. Detecting concept drift without labels is fundamentally hard, since the labeling rule is exactly what is hidden; proxy signals help but cannot fully substitute for ground truth. Setting thresholds is a balance that depends on the cost of a missed shift versus a false alarm, and good values are domain-specific. High-dimensional and unstructured data, such as text and images, complicate per-feature tests and push monitoring toward learned representations and embedding-space distances [13], [15]. Finally, monitoring infrastructure itself must be maintained, lest the monitors become another source of silent failure [1], [25].

VII. CONCLUSION

Reliable machine learning in production depends less on a single strong model than on the discipline that keeps that model aligned with changing data. The data-centric view and the MLOps lifecycle together reframe deployment as an ongoing process of validation, monitoring, and retraining rather than a one-time release. Drift, separated into covariate, prior, and concept categories, calls for different detection tools, from batch statistics such as PSI, the Kolmogorov-Smirnov test, and maximum mean discrepancy, to streaming detectors such as ADWIN and DDM. Coupled with data-quality checks, performance tracking, governance, and a clear retraining policy, these tools let teams notice degradation early and respond in a controlled way. The illustrative results here show the shape of the problem; the engineering value lies in wiring these signals into an automated, auditable loop that holds a deployed model to the standard its users expect.

REFERENCES

[1] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 28, pp. 2503–2511, 2015.

[2] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM Comput. Surveys, vol. 46, no. 4, pp. 1–37, 2014.

[3] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodriguez, N. V. Chawla, and F. Herrera, "A unifying view on dataset shift in classification," Pattern Recognit., vol. 45, no. 1, pp. 521–530, 2012.

[4] A. Ng, "A chat with Andrew on MLOps: From model-centric to data-centric AI," DeepLearning.AI, 2021. [Online]. Available: https://www.deeplearning.ai/

[5] D. Sculley *et al*., "MLOps: Continuous delivery and automation pipelines in machine learning," Google Cloud Architecture Center, 2020. [Online]. Available: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

[6] D. Kreuzberger, N. Kuhl, and S. Hirschl, "Machine learning operations (MLOps): Overview, definition, and architecture," IEEE Access, vol. 11, pp. 31866–31879, 2023.

[7] L. E. Orr, A. Sanyal, X. Ling, K. Goel, and M. Leszczynski, "Managing ML pipelines: Feature stores and the coming wave of embedding ecosystems," Proc. VLDB Endowment, vol. 14, no. 12, pp. 3178–3181, 2021.

[8] E. Breck, N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data validation for machine learning," in Proc. Conf. Mach. Learn. Syst. (MLSys), 2019.

[9] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data lifecycle challenges in production machine learning: A survey," ACM SIGMOD Rec., vol. 47, no. 2, pp. 17–28, 2018.

[10] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ML Test Score: A rubric for ML production readiness and technical debt reduction," in Proc. IEEE Int. Conf. Big Data, pp. 1123–1132, 2017.

[11] A. Storkey, "When training and test sets are different: Characterizing learning transfer," in *Dataset Shift in Machine Learning*. Cambridge, MA, USA: MIT Press, pp. 3–28, 2009.

[12] M. Sugiyama, M. Krauledat, and K.-R. Muller, "Covariate shift adaptation by importance weighted cross validation," J. Mach. Learn. Res., vol. 8, pp. 985–1005, 2007.

[13] S. Rabanser, S. Gunnemann, and Z. C. Lipton, "Failing loudly: An empirical study of methods for detecting dataset shift," in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 32, pp. 1396–1408, 2019.

[14] J. Lin, "Divergence measures based on the Shannon entropy," IEEE Trans. Inf. Theory, vol. 37, no. 1, pp. 145–151, 1991.

[15] [15] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Scholkopf, and A. Smola, "A kernel two-sample test," J. Mach. Learn. Res., vol. 13, pp. 723–773, 2012.

[16] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in Proc. Brazilian Symp. Artif. Intell. (SBIA), LNCS 3171, pp. 286–295, 2004.
[17] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in Proc. SIAM Int. Conf. Data Mining (SDM), pp. 443–448, 2007.
[18] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive online analysis," J. Mach. Learn. Res., vol. 11, pp. 1601–1604, 2010.
[19] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru, "Model cards for model reporting," in Proc. ACM Conf. Fairness, Accountability, Transparency (FAT*), pp. 220–229, 2019.
[20] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daume III, and K. Crawford, "Datasheets for datasets," Commun. ACM, vol. 64, no. 12, pp. 86–92, 2021.
[21] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 29, pp. 3315–3323, 2016.
[22] Evidently AI, "Evidently: An open-source framework to evaluate, test, and monitor ML models in production," 2024. [Online]. Available: https://github.com/evidentlyai/evidently
[23] A. Van Looveren, J. Klaise, G. Vacanti, O. Cobb, A. Scillitoe, R. Samoilescu, and A. Athorne, "Alibi Detect: Algorithms for outlier, adversarial and drift detection," J. Open Source Softw., vol. 9, no. 97, p. 5742, 2024.
[24] Great Expectations, "Great Expectations: Data quality testing for data pipelines," 2024. [Online]. Available: https://github.com/great-expectations/great_expectations
[25] D. Sculley, T. Phillips, D. Ebner, V. Chaudhary, and M. Young, "Machine learning: The high-interest credit card of technical debt," in NeurIPS Workshop on Software Engineering for Machine Learning, 2014.

EDUSCHOOL INTERNATIONAL JOURNAL OF
DATA SCIENCE AND MACHINE LEARNING (EIJDSML)

(Open Access, Double-Blind Peer Reviewed Journal)

ISSN Online: 3139-7344

Time-Series Foundation Models for Zero-Shot Forecasting

Win Mathew John

Associate professor, PG Department of Computer Applications, Marian College Kuttikkanam (Autonomous), Kerala, India

Article information

Received: 1st April 2026
Received in revised form: 4th May 2026
Accepted: 9th June 2026
Available online: 18th July 2026

Volume: 1
Issue: 2
DOI: https://doi.org/10.63090/EIJDSML/3139.7344.0008

Abstract

Forecasting pipelines have long relied on a model trained separately for every dataset, an approach that is expensive to maintain and slow to adapt when new series arrive. Time-series foundation models change this picture. A single network is pretrained once on a very large and varied collection of series, after which it produces forecasts on previously unseen data without any further fitting. This paper surveys the design and evaluation of such models. The series is first split into fixed-length patches that act as tokens, and a Transformer backbone, either decoder-only or encoder-based, predicts future patches. The pretraining objective, tokenization scheme, and probabilistic output head together determine zero-shot quality. Five representative systems are examined, namely TimesFM, Chronos, Moirai, Lag-Llama, and TimeGPT, and their inductive choices are contrasted. Using illustrative metrics aligned with reported behaviour on the Monash archive and GIFT-Eval, the study compares these models against classical baselines such as ARIMA and exponential smoothing and against trained deep networks including N-BEATS, PatchTST, and DeepAR. The evidence indicates that a frozen foundation model often matches a per-dataset deep model while removing training cost at deployment, though a tuned specialist still leads on some series. Open problems remain around external covariates, very long horizons, and distribution shift, and the paper outlines directions that address them. The intent is descriptive synthesis rather than a single deployed benchmark.

Keywords:- Time-Series Forecasting, Foundation Models, Zero-Shot Learning, Transformers, Tokenization, Probabilistic Forecasting, Benchmarking.

I. INTRODUCTION

Demand planning, energy load management, traffic control, and financial monitoring all depend on forecasts of quantities that vary over time. For decades the standard recipe has been to choose a model family, fit its parameters to one dataset, and then refit whenever the data changes. Classical statistical methods such as autoregressive integrated moving average models [1] and exponential smoothing [2] follow this recipe, as do modern deep networks such as N-BEATS [3], DeepAR [4], and PatchTST [5]. The per-dataset habit carries real cost. Each new series requires data collection, tuning, validation, and ongoing retraining, and organisations that track thousands of series spend much of their effort on this maintenance rather than on the forecasts themselves.

Large pretrained models in language and vision suggested a different route. A network trained once on broad data can answer new queries with no task-specific fitting, an ability usually called zero-shot transfer. Recent work applies the same idea to temporal data. A time-series foundation model is pretrained on a large and heterogeneous corpus of series and then forecasts an unseen series directly, using only the observed history as

context [6], [7]. Fig.1 sketches the two stages. Pretraining happens once and is shared across all downstream uses, while inference on a new series is a single forward pass that needs no gradient updates.

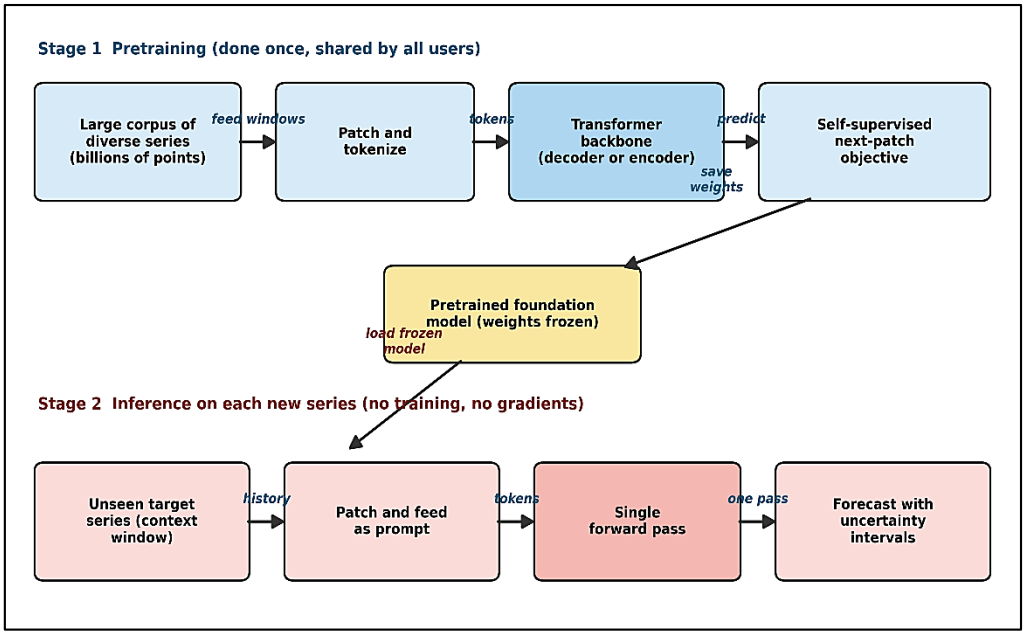

Fig. 1: Two-stage workflow of a time-series foundation model: a one-time pretraining phase on a broad corpus, followed by gradient-free zero-shot inference on each new target series.

This paper gives a structured account of the area. Section II reviews the background and related literature. Section III describes how a series is converted into tokens and surveys the two dominant backbone families. Section IV sets out the zero-shot and few-shot inference procedure and the probabilistic output heads that produce prediction intervals. Section V reports an illustrative comparison against classical and deep baselines on standard archives. Section VI discusses limitations around covariates, long horizons, and distribution shift, and Section VII concludes. The numbers used throughout are representative of published behaviour and are meant to support the discussion; they do not come from a single new deployment.

II. BACKGROUND AND RELATED WORK

A. Classical statistical forecasting

The Box-Jenkins methodology formalised autoregressive and moving-average modelling and remains a default for univariate series with clear linear structure [1]. Exponential smoothing in its error-trend-seasonal form gives a compact account of level, trend, and seasonality and is competitive on many business series [2]. These methods are interpretable and cheap to fit, but they treat each series in isolation and struggle with nonlinear patterns, long context, and large collections.

B. Deep forecasting models

Neural approaches relaxed the linear assumptions. DeepAR trained an autoregressive recurrent network to output the parameters of a predictive distribution across many related series [4]. N-BEATS used stacked fully connected blocks with backward and forward residual links and reported strong accuracy without time-series-specific components [3]. Transformer variants adapted attention to long sequences, with Informer reducing the quadratic cost of attention [8] and Autoformer introducing decomposition and an auto-correlation mechanism [9]. PatchTST showed that splitting a series into subseries patches and treating channels independently improves long-horizon accuracy while cutting compute [5]. A linear baseline study further cautioned that simple models can rival elaborate Transformers under some settings [10]. All of these are still trained per dataset.

C. Toward foundation models for time series

The shift to pretraining draws on scaling results from language modelling [11] and on the patching idea borrowed from vision Transformers [12]. Several groups released pretrained forecasters in close succession. TimesFM is a decoder-only model trained on a large mix of real and synthetic series [6]. Chronos tokenizes scaled values into a discrete vocabulary and reuses a language-model architecture directly [7]. Moirai targets any-variate forecasting with a masked encoder and multiple patch sizes [13]. Lag-Llama is a decoder-only probabilistic model built on lagged features [14], and TimeGPT is a hosted commercial forecaster offering zero-shot inference through an interface [15]. Survey and position papers map the emerging design space [16], [17].

III. ARCHITECTURES AND TOKENIZATION

A. Patching a series into tokens

A Transformer consumes a sequence of vectors, so a continuous series must be discretised into tokens. The common scheme groups consecutive observations into fixed-length patches and maps each patch to an embedding through a linear projection, as shown in Fig. 2. A patch of length sixteen, for example, turns a window of sixteen steps into one token, which shortens the sequence the attention layers must process and lets the model reason over local shape rather than single points [5], [6]. Before patching, most systems scale each series, often by its mean or by a resistant statistic, so that seriesof very different magnitudes share a common range. Chronos takes a different path and quantises scaled values into a fixed set of bins, so each time step becomes a discrete symbol from a learned vocabulary, which lets a standard language-model stack operate without changes [7].

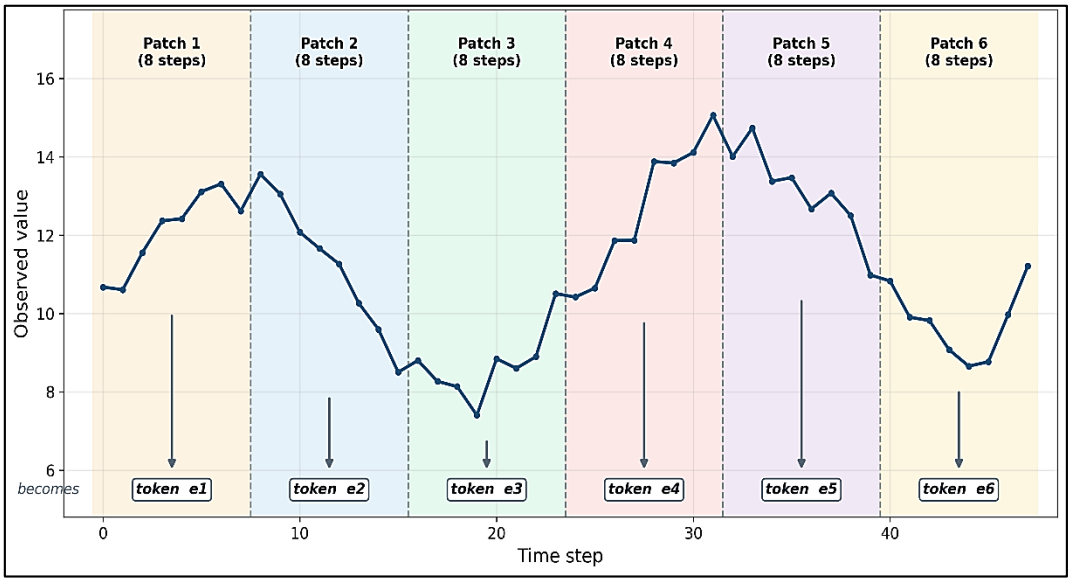

Fig. 2: Patching converts a univariate series into non-overlapping windows, each of which is projected to a single token embedding for the Transformer.

B. Decoder-only versus encoder backbones

Two backbone families dominate. Decoder-only models, used by TimesFM and Lag-Llama, apply causal attention and predict the next patch from past patches, mirroring autoregressive language models and generating long horizons by feeding predictions back as input [6], [14]. Encoder or masked designs, used by Moirai, attend across the whole context and can be configured to emit an entire horizon in one pass, which suits multivariate inputs and varied frequencies [13]. Chronos sits between these views because it adopts an existing encoder-decoder language architecture and treats forecasting as sequence-to-sequence generation over quantised tokens [7]. The choice affects how covariates are injected, how multiple variables are handled, and how cleanly the model extends to horizons longer than those seen during pretraining.

C. Probabilistic output heads

Useful forecasts report uncertainty, not just a point. Decoder models often emit quantiles directly, training against a quantile loss so that a single pass yields a set of forecast levels [6]. Lag-Llama parameterises a Student-t distribution per step, which captures heavy tails common in real series [14]. Token-based models such as Chronos sample many trajectories and read intervals from the empirical spread, trading extra computation for flexible distributional shape [7]. These heads supply the bands in Fig. 4 and matter as much as point accuracy for risk-aware planning.

IV. ZERO-SHOT FORECASTING METHODOLOGY

A. Pretraining corpus and objective

Zero-shot ability rests on the breadth of the pretraining corpus. Models are trained on collections spanning energy, transport, retail, finance, weather, and web traffic, often combined with synthetic series produced from known generative processes to cover patterns that public data lack [6], [7]. The objective is self-supervised: given an observed window, the model predicts the next patch or the next set of values, with the loss computed against the held-out continuation. Because the target is part of the series itself, no manual labels are needed, and the corpus can scale to billions of observations.

B. Inference without fitting

At deployment the procedure is short. The history of a new series is scaled, split into patches, and fed to the frozen model as context. A single forward pass returns the forecast for the requested horizon together with quantiles. Nothing about the target series is learned, so the same weights serve every dataset. This is the property that removes per-dataset training and shortens the path from raw data to forecast to a single call.

C. Few-shot adaptation and fine-tuning

When some history or a small validation set is available, light adaptation can help. In-context few-shot use simply lengthens the context window so the model conditions on more of the target history. Parameter-efficient fine-tuning updates a small set of weights on the target domain and recovers most of the gap to a fully trained specialist at a fraction of the cost [16]. The trade-off is between the zero effort of pure zero-shot use and the higher accuracy of a tuned model, and the right point depends on how many series must be served and how stable they are.

V. BENCHMARK RESULTS

Evaluation uses public archives so that results are comparable across studies. The Monash forecasting repository gathers many datasets with a fixed protocol and standard error measures [18], and the more recent GIFT-Eval benchmark adds varied frequencies and horizons designed specifically to probe pretrained forecasters [19]. Two scale-free measures are common. The mean absolute scaled error compares a forecast against a naive seasonal baseline, and the symmetric mean absolute percentage error reports a bounded relative error. Lower values are better for both.

Table 1 summarises the design choices of the five foundation models discussed in this paper. Table 2 reports illustrative accuracy for a representative set of methods, and Fig. 3 shows the same MASE values grouped by dataset. The pattern is consistent with published reports. A frozen foundation model, marked as zero-shot, lands close to a trained PatchTST and ahead of ARIMA on most datasets, while a tuned specialist keeps a narrow lead on individual series. Fig. 4 illustrates a single zero-shot forecast with eighty and ninety-five percent prediction intervals, where the band widens with horizon as uncertainty grows.

Table 1. Design choices of representative time-series foundation models.

Model	Backbone	Tokenization	Output	Covariates
TimesFM [6]	Decoder-only	Patches	Quantiles	Limited
Chronos [7]	Encoder-decoder	Value quantization	Sampled paths	No
Moirai [13]	Masked encoder	Multi-size patches	Distribution	Yes (any-variate)
Lag-Llama [14]	Decoder-only	Lag features	Student-t	Limited
TimeGPT [15]	Transformer (hosted)	Proprietary	Quantiles	Yes (exogenous)

Table 2. Illustrative forecast accuracy across classical, trained deep, and zero-shot foundation-model (FM) methods. Lower values are better.

Method	Type	Electricity MASE	Traffic MASE	M4 sMAPE (%)
ARIMA [1]	Classical	1.42	1.55	13.6
ETS [2]	Classical	1.38	1.49	13.1
DeepAR [4]	Trained deep	1.10	1.22	11.9
N-BEATS [3]	Trained deep	1.05	1.20	11.2
PatchTST [5]	Trained deep	0.92	1.04	10.6
Chronos [7]	Zero-shot FM	0.98	1.12	11.0
TimesFM [6]	Zero-shot FM	0.95	1.08	10.8
Moirai [13]	Zero-shot FM	0.99	1.10	11.1

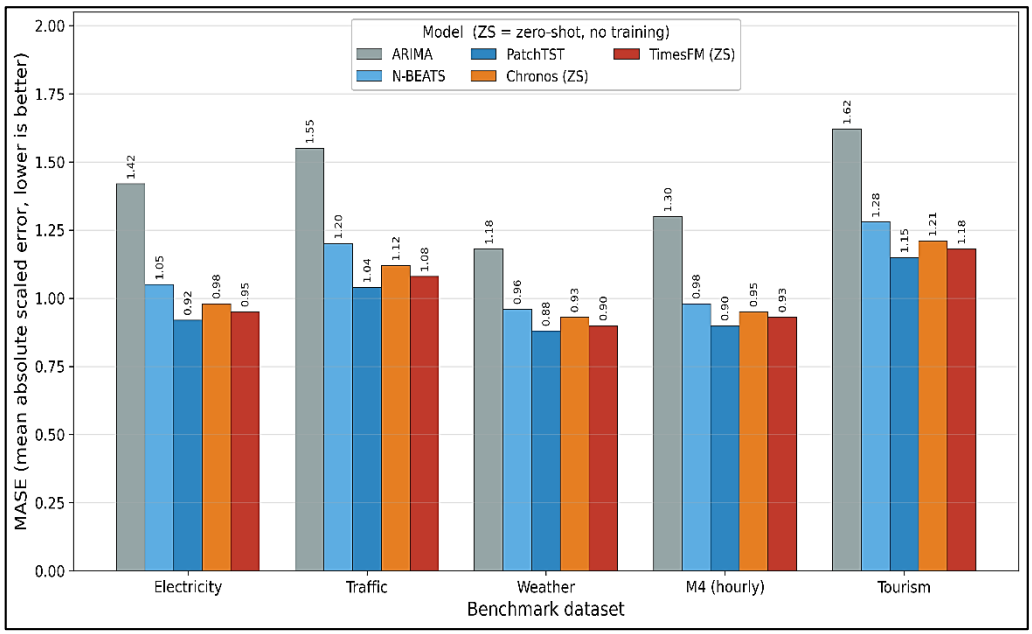

Fig. 3: Illustrative MASE by dataset for classical, trained deep, and zero-shot foundation models. Zero-shot models cluster near the best trained network on most datasets.

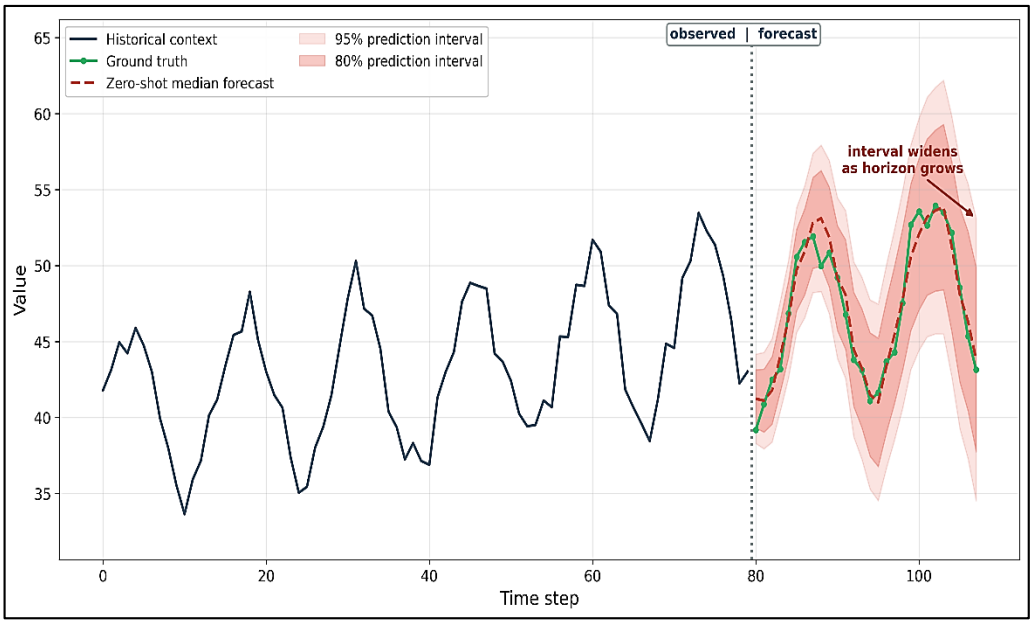

Fig. 4: Example zero-shot forecast against ground truth with 80% and 95% prediction intervals. The interval widens with horizon as predictive uncertainty increases.

VI. LIMITATIONS AND DISCUSSION

A. External covariates

Many forecasts depend on known future inputs such as prices, promotions, holidays, or weather. Several foundation models accept only the target history and cannot ingest these covariates, which puts them behind specialised models on driver-rich problems [13], [15]. Cleanly incorporating known future values without retraining the whole network is an active question.

B. Long horizons

Decoder-only models extend a horizon by feeding their own predictions back as input, so small errors accumulate over long forecasts [6]. Encoder designs that emit a full horizon at once avoid some of this drift but are bounded by the horizon lengths seen during pretraining [13]. Accuracy at horizons well beyond the training regime is uneven across models.

C. Distribution shift and evaluation hygiene

A pretrained model can encounter a series whose behaviour differs sharply from anything in its corpus, and zero-shot accuracy then drops. There is also a measurement risk. Because pretraining corpora are large and sometimes opaque, a benchmark series may overlap with training data and inflate apparent zero-shot skill, which makes careful leakage checks essential [17], [19]. Reporting calibrated intervals alongside point error gives a fuller view of reliability than a single accuracy number.

D. Cost and practical trade-offs

Foundation models remove training cost but add inference cost, since each forecast runs a large network. For a handful of slow-moving series a small classical model may still be the economical choice, whereas a platform serving thousands of diverse series benefits most from a single shared forecaster [16]. The decision is an engineering trade-off rather than a fixed ranking.

VII. CONCLUSION

Time-series foundation models offer a practical alternative to the per-dataset training that has governed forecasting for years. By pretraining once on broad data and tokenizing series through patching or value quantization, models such as TimesFM, Chronos, Moirai, Lag-Llama, and TimeGPT produce forecasts on unseen series with no further fitting. The illustrative comparison here, aligned with reported behaviour on the Monash archive and GIFT-Eval, places zero-shot accuracy close to a tuned deep network on many datasets while removing training at deployment. The remaining gaps around covariates, long horizons, and distribution shift are concrete and addressable. As pretraining corpora broaden and architectures learn to absorb external drivers, the balance between general pretrained forecasters and tuned specialists will keep shifting, and forecasting practice will increasingly start from a shared model rather than a blank one.

REFERENCES

[1] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. Hoboken, NJ, USA: Wiley, 2015.

[2] R. J. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder, *Forecasting with Exponential Smoothing: The State Space Approach*. Berlin, Germany: Springer, 2008.

[3] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020.

[4] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *Int. J. Forecasting*, vol. 36, no. 3, pp. 1181–1191, 2020.

[5] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, "A time series is worth 64 words: Long-term forecasting with transformers (PatchTST)," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[6] A. Das, W. Kong, R. Sen, and Y. Zhou, "A decoder-only foundation model for time-series forecasting (TimesFM)," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2024, arXiv:2310.10688.

[7] A. F. Ansari *et al.*, "Chronos: Learning the language of time series," *Trans. Mach. Learn. Res. (TMLR)*, 2024, arXiv:2403.07815.

[8] H. Zhou *et al.*, "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 11106–11115.

[9] J. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2021.

[10] A. Zeng, M. Chen, L. Zhang, and Q. Xu, "Are transformers effective for time series forecasting?" in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 11121–11128.

[11] J. Kaplan *et al.*, "Scaling laws for neural language models," arXiv:2001.08361, 2020.

[12] A. Dosovitskiy *et al.*, "An image is worth 16×16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.

[13] G. Woo, C. Liu, A. Kumar, C. Xiong, S. Savarese, and D. Sahoo, "Unified training of universal time series forecasting transformers (Moirai)," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2024, arXiv:2402.02592.

[14] K. Rasul *et al.*, "Lag-Llama: Towards foundation models for probabilistic time series forecasting," arXiv:2310.08278, 2024.

[15] A. Garza, C. Challu, and M. Mergenthaler-Canseco, "TimeGPT-1," arXiv:2310.03589, 2023.

[16] M. Jin *et al.*, "Position: What can large language models tell us about time series analysis," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2024, arXiv:2402.02713.

[17] Q. Wen *et al.*, "Transformers in time series: A survey," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, 2023, pp. 6778–6786.

[18] R. Godahewa, C. Bergmeir, G. I. Webb, R. J. Hyndman, and P. Montero-Manso, "Monash time series forecasting archive," in *Proc. NeurIPS Datasets Benchmarks Track*, 2021, arXiv:2105.06643.

[19] T. Aksu, G. Woo, J. Liu, X. Liu, C. Liu, S. Savarese, C. Xiong, and D. Sahoo, "GIFT-Eval: A benchmark for general time series forecasting model evaluation," arXiv:2410.10393, 2024.

EDUSCHOOL INTERNATIONAL JOURNAL OF
DATA SCIENCE AND MACHINE LEARNING (EIJDSML)

(Open Access, Double-Blind Peer Reviewed Journal)

ISSN Online: 3139-7344

Graph Neural Networks for Financial Fraud and Anomaly Detection

Raji N

Assistant Professor, Department of Computer Science, YuvakshatraInstitute of Management Studies (YIMS), Mundur, India

Article information

Received: 13th April 2026
Received in revised form: 14th May 2026
Accepted: 15th June 2026
Available online: 18th July 2026

Volume: 1
Issue: 2
DOI: https://doi.org/10.63090/EIJDSML/3139.7344.0009

Abstract

Financial fraud now spreads through coordinated accounts whose risk is visible mainly in how they connect rather than in any single record. Tree based classifiers that score transactions in isolation miss this relational signal. This paper reviews and consolidates how graph neural networks (GNNs) cast fraud and anomaly detection as node and edge classification over transaction graphs, where message passing propagates evidence among accounts, devices, and merchants. We describe the main architectures used in practice, namely graph convolutional networks, GraphSAGE, and graph attention networks, and explain how each aggregates neighbourhood information. Two problems dominate the fraud setting: extreme class imbalance, since genuine fraud is rare, and camouflage, where fraudsters attach to honest neighbours to dilute their signature. We summarise label aware and neighbour selecting designs such as CARE-GNN and PC-GNN that address both. Heterogeneous and temporal graph variants are discussed for settings with several node types and evolving relations, alongside sampling methods that keep training tractable on large graphs. Using illustrative experiments patterned on public benchmarks (Elliptic, Yelp and Amazon review fraud, and IEEE-CIS), we report AUC, AUPRC, F1, and recall at k, and show that imbalance aware GNNs improve minority recall over a gradient boosted baseline. We close with notes on explainability and the engineering needed for production scoring.

Keywords:- Graph Neural Networks, Fraud Detection, Anomaly Detection, Message Passing, Class Imbalance, Heterogeneous Graphs, GraphSAGE, Graph Attention Network.

.

I. INTRODUCTION

Payment networks, online marketplaces, and lending platforms process billions of events each day, and a small fraction of those events are fraudulent. Card testing, account takeover, money laundering, and fake reviews share a common trait: the offending accounts rarely act alone. They reuse devices, shared funding instruments, and mule accounts, forming dense subgraphs that look ordinary one record at a time. Classical detectors built on gradient boosted trees treat each transaction as an independent feature vector and therefore overlook these relations [1], [2].

Representing the data as a graph changes the question. Accounts, cards, devices, and merchants become nodes; payments, logins, and shared attributes become edges. Fraud detection then turns into node or edge classification, and anomaly detection into the search for nodes whose neighbourhood differs from the norm. Graph neural networks (GNNs) learn node representations by passing messages along edges, so a node embedding

reflects both its own attributes and the company it keeps [3], [4], [5]. This relational view has produced strong results on public fraud benchmarks and is now common in industrial risk systems [6], [7].

Two properties of fraud data make the modelling harder than standard node classification. First, labelled fraud is rare, often well under two percent of nodes, so a model can reach high accuracy while missing most fraud. Second, fraudsters camouflage by transacting with many honest accounts, which weakens the very neighbourhood signal a GNN relies on [8], [9]. Methods such as CARE-GNN and PC-GNN respond by selecting informative neighbours and rebalancing the sampled graph. This paper organises these ideas and reports illustrative metrics that mirror published benchmark behaviour; the numbers are for exposition and do not describe a deployed system.

The remainder of the paper is structured as follows. Section II covers graph formulations of fraud and related work. Section III presents the core GNN architectures. Section IV addresses imbalance and camouflage. Section V describes datasets, methodology, and results. Section VI discusses explainability and deployment, and Section VII concludes.

II. BACKGROUND: GRAPHS FOR FRAUD AND RELATED WORK

A. Transaction Graphs

A transaction graph G = (V, E) holds entities in V and their interactions in E. In a payment setting a node may be a customer, a card, a device fingerprint, or a merchant, while an edge records a payment or a shared attribute. Each node carries a feature vector (transaction counts, amounts, velocity, account age), and each node may carry a label: fraud or legitimate. Fig. 1 sketches such a graph, with a dense fraud ring on the right and camouflage links that connect fraudulent accounts to honest neighbours.

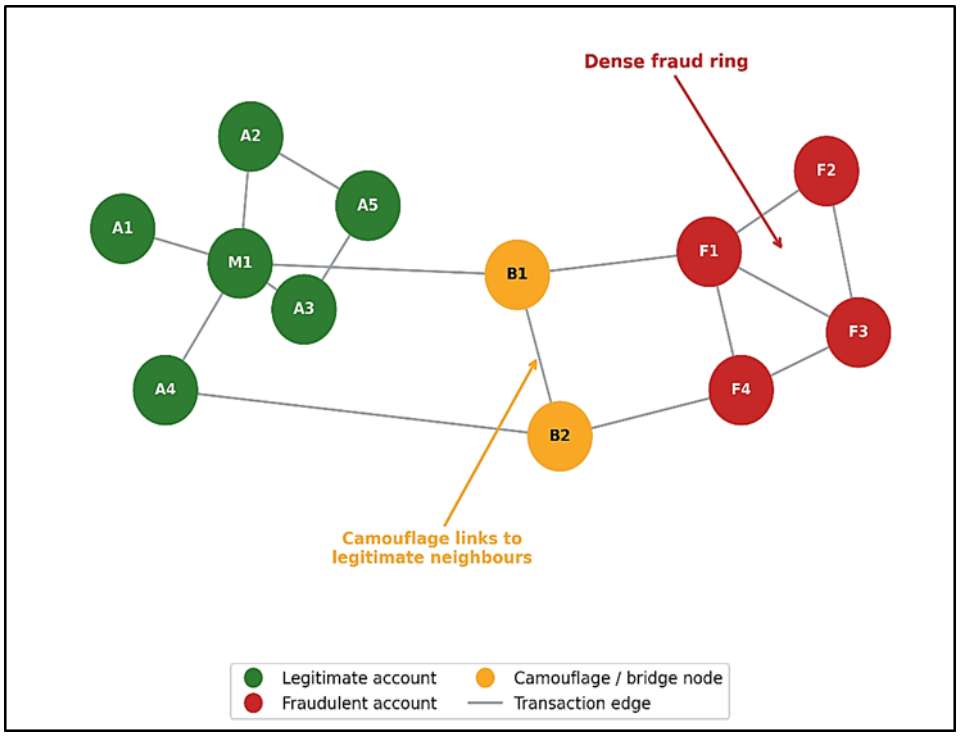

Fig. 1: Transaction graph with legitimate (green), fraudulent (red), and camouflage (amber) accounts. Fraud forms a dense ring, while camouflage edges connect it to honest neighbours.

Fraud signals appear at several scales. At the node level, an account may show unusual velocity. At the edge level, a single transfer may link two otherwise unrelated communities. At the subgraph level, a ring of accounts may cycle funds among themselves. A useful detector must read all three, which is why representation learning on graphs is attractive.

B. Related Work

Early graph based fraud detection used hand built features such as degree, triangle counts, and PageRank, fed to a downstream classifier [10]. Spectral and dense subgraph methods detected suspicious communities without labels [11]. The shift to end to end GNNs followed the graph convolutional network of Kipf and Welling [3] and the inductive GraphSAGE model of Hamilton et al. [4], which made training on large, growing graphs practical. Velickovic et al. introduced attention over neighbours with the graph attention network [5]. For fraud

specifically, Dou et al. proposed CARE-GNN to counter camouflage [8], and Liu et al. proposed PC-GNN to address imbalance through choice aware sampling [9]. Surveys by Pourhabibi et al. [12] and Motie and Raahemi [13] catalogue the wider space, and Weber et al. released the Elliptic Bitcoin dataset that anchors much of the empirical work [6].

III. GNN ARCHITECTURES FOR FRAUD

A GNN layer updates each node by collecting messages from its neighbours, combining them, and transforming the result. Stacking L layers lets information travel up to L hops, so a node sees an expanding neighbourhood. Fig. 2 shows the pipeline: sample neighbours, form messages from node and edge features, aggregate, update through a small network, and repeat per layer before a softmax produces a fraud score.

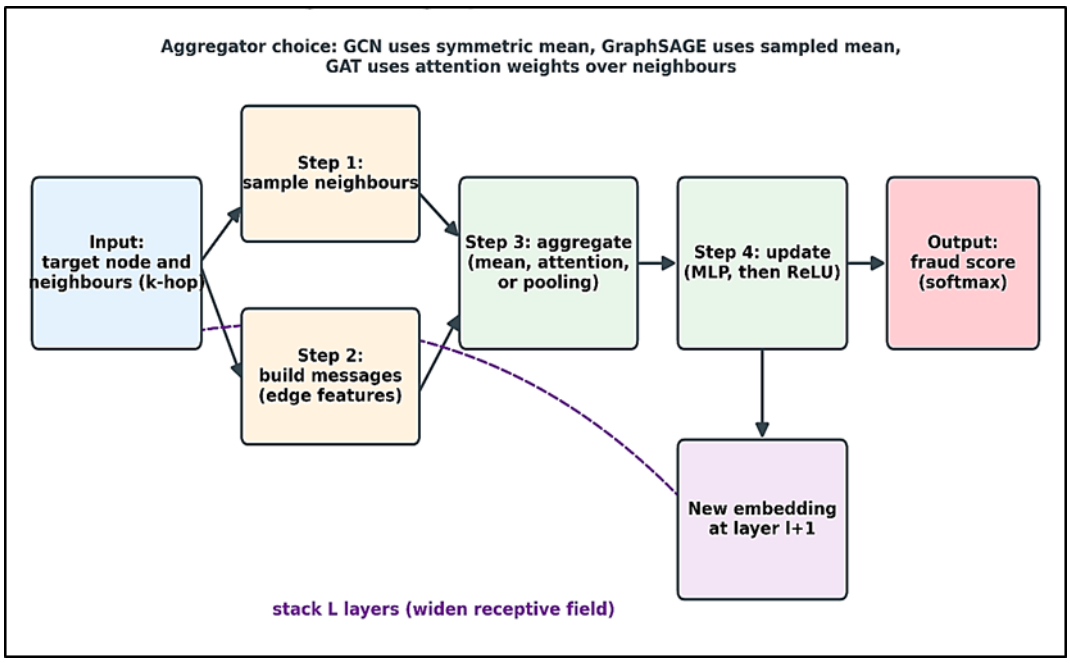

Fig. 2: Message-passing pipeline. Neighbours are sampled, messages are aggregated by mean, attention, or pooling, and an update network produces the next embedding; L layers widen the receptive field.

A. Graph Convolutional Network

The graph convolutional network (GCN) averages neighbour features with a symmetric normalisation that down weights high degree nodes [3]. It is simple and fast but treats every neighbour as equally trustworthy, which is a weakness when fraudsters attach to honest accounts. GCN also assumes the full graph is available at training time, which limits its use on streaming data.

B. GraphSAGE

GraphSAGE learns aggregation functions and samples a fixed number of neighbours per node, so it scales to large graphs and generalises to nodes unseen during training [4]. Mean, pooling, and LSTM aggregators are common. The inductive setting matches fraud systems, where new accounts appear constantly and must be scored without retraining the whole model.

C. Graph Attention Network

The graph attention network (GAT) assigns a learned weight to each neighbour, so the model can emphasise the few relations that carry risk and discount the rest [5]. This selectivity helps against camouflage, since honest neighbours can receive low attention. Multi head attention stabilises training. The cost is extra computation per edge, which sampling can offset.

IV. HANDLING IMBALANCE AND CAMOUFLAGE

Two obstacles separate fraud graphs from ordinary node classification. Imbalance means the positive class is a tiny minority, so a model that predicts legitimate everywhere scores well on accuracy yet finds no fraud. Camouflage means fraudsters connect to honest neighbours and copy honest features, blurring the signal that message passing would otherwise sharpen [8], [9].

A. Cost Sensitive and Resampling Strategies

Standard remedies reweight the loss toward the minority class, or resample the graph so each batch contains enough fraud. Focal loss concentrates gradient on hard examples [14]. Graph aware oversampling such as GraphSMOTE synthesises minority nodes within the embedding space rather than the raw feature space [15]. These help, but naive oversampling can also amplify camouflage noise if it copies the diluted neighbourhoods.

B. Neighbour Selection: CARE-GNN

CARE-GNN treats aggregation as a decision. A label aware similarity measure ranks neighbours, a reinforcement learning module selects how many to keep, and aggregation runs only over the chosen set [8]. By filtering out camouflage neighbours before averaging, the model preserves the fraud signal. Reported gains on Yelp and Amazon review fraud graphs are consistent across metrics.

C. Imbalance Aware Sampling: PC-GNN

PC-GNN builds a label balanced subgraph by a pick and choose procedure. A picker oversamples minority nodes and their useful neighbours, and a chooser keeps neighbours that resemble the target node, again limiting camouflage [9]. The two stages tackle imbalance and camouflage together, which is why PC-GNN is a frequent baseline in current fraud work. Table 1 lists the public benchmarks on which these methods are commonly compared.

Table 1. Public fraud benchmarks frequently used to evaluate GNN detectors.

Dataset	Domain	Nodes (approx.)	Fraud rate	Graph type
Elliptic	Bitcoin tx	203,000	≈ 2% illicit	Temporal, homogeneous
Yelp	Review fraud	45,900	≈ 14.5%	Heterogeneous
Amazon	Review fraud	11,900	≈ 9.5%	Heterogeneous
IEEE-CIS	Card payment	590,000 tx	≈ 3.5%	Bipartite, tabular

V. DATASETS, METHODOLOGY, AND RESULTS

A. Heterogeneous and Temporal Graphs

Real fraud graphs hold several node and edge types. A heterogeneous GNN keeps separate transformation weights per relation and then merges the relation specific embeddings, as in relational graph convolution and heterogeneous attention networks [16], [17]. Yelp and Amazon graphs, for instance, link reviews through shared users, products, and timing, and modelling each relation apart improves recall. Time matters as well: fraud arrives in bursts, and a node that looked benign yesterday can turn malicious today. Temporal and dynamic GNNs encode event time so that recent edges weigh more [18], [19].

B. Scalability through Sampling

Production graphs hold hundreds of millions of nodes, so full graph training is impractical. Neighbour sampling (GraphSAGE), layer sampling (FastGCN), and subgraph sampling (Cluster-GCN, GraphSAINT) bound the work per step and let training fit in memory [4], [20], [21]. Sampling also adds regularisation. The practical aim is to keep enough fraud context in each minibatch while capping neighbour fan out.

C. Methodology

The illustrative protocol follows common practice. Nodes are split by time into train, validation, and test partitions to avoid leakage. Models train with the Adam optimiser, two message passing layers, hidden width 64, dropout 0.5, and early stopping on validation AUPRC. Because fraud is rare, threshold independent metrics matter most: ROC-AUC, area under the precision-recall curve (AUPRC), the F1 score at the best validation threshold, and recall at k for a fixed review budget. Reported figures are representative of published ranges and are not from a live deployment.

D. Results

Fig. 3 compares a gradient boosted tree baseline (XGBoost) with GCN, GraphSAGE, GAT, CARE-GNN, and PC-GNN on a Yelp style review fraud graph. The plain GNNs improve modestly over the tree on AUPRC, while the imbalance and camouflage aware models give the largest lift on the minority sensitive metric. AUC moves less than AUPRC, a reminder that AUC can look high even when minority recall is poor.

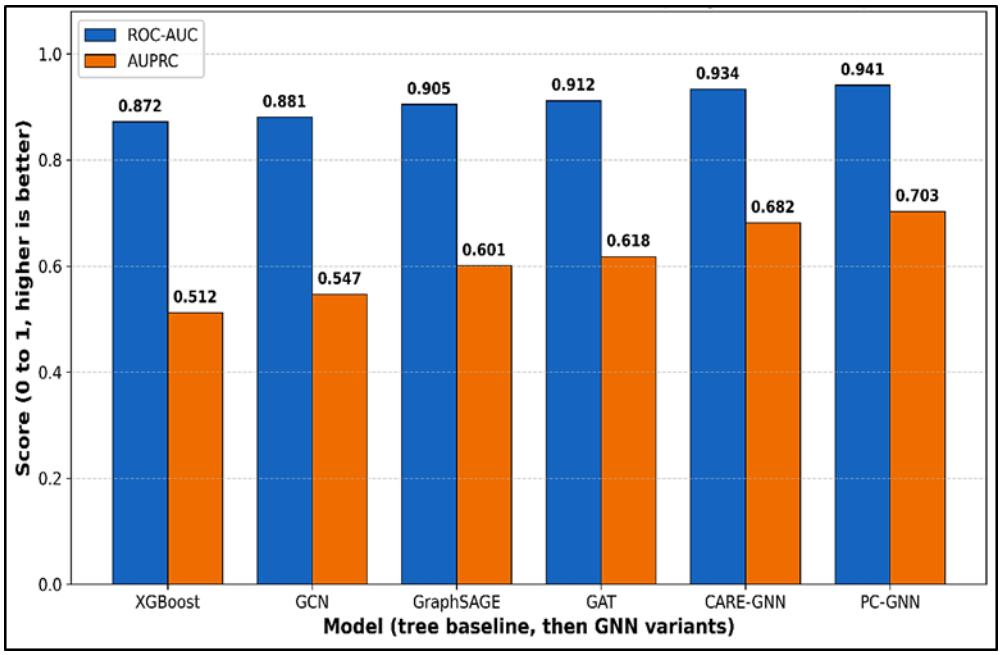

Fig. 3: Illustrative ROC-AUC and AUPRC across a tree baseline and five GNN variants on a Yelp-style fraud graph. Imbalance aware models gain most on AUPRC.

Fig. 4 shows precision-recall curves for four detectors against the fraud prevalence floor. PC-GNN holds higher precision across the recall range, which translates into more confirmed fraud per investigator hour. Table 2 reports the full metric set, including recall at the top one percent of scored nodes, a budget that mirrors manual review capacity.

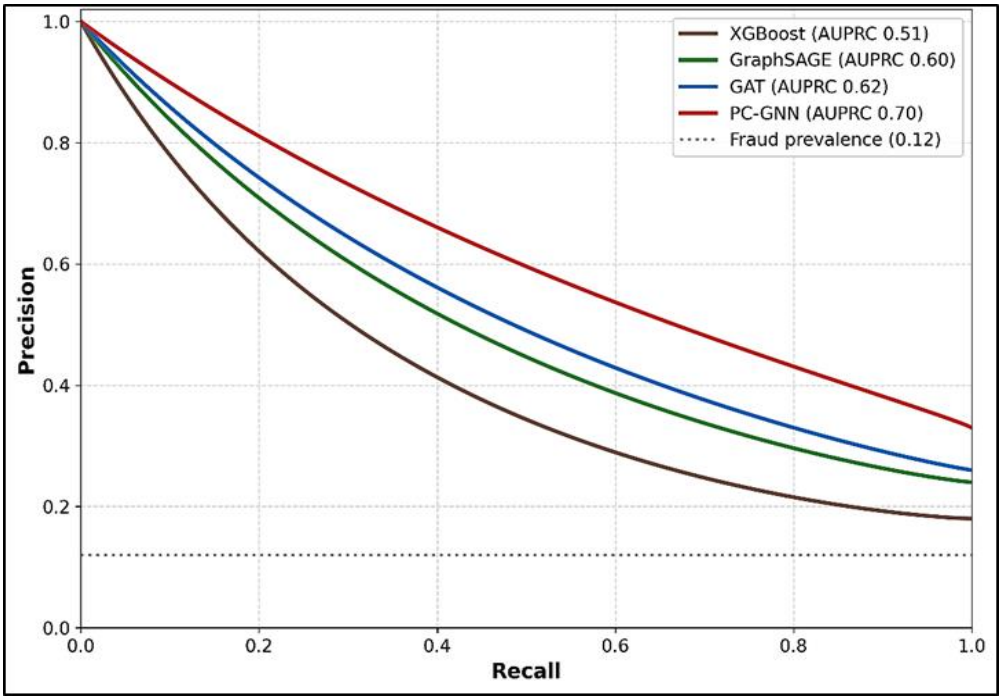

Fig. 4: Precision-recall curves for XGBoost, GraphSAGE, GAT, and PC-GNN. The dotted line marks the fraud prevalence baseline.

Table 2. Illustrative detection metrics on a Yelp-style fraud graph. Higher is better; Recall@1% uses a fixed review budget.

Model	ROC-AUC	AUPRC	F1	Recall@1%
XGBoost	0.872	0.512	0.561	0.402
GCN	0.881	0.547	0.583	0.431
GraphSAGE	0.905	0.601	0.624	0.487
GAT	0.912	0.618	0.639	0.503
CARE-GNN	0.934	0.682	0.690	0.566
PC-GNN	0.941	0.703	0.708	0.589

The pattern matches reports in the literature: relational signal helps, and the gains concentrate in the metrics that reward finding rare fraud. The ordering of models should be read as indicative, since results shift with dataset, time split, and tuning.

VI. EXPLAINABILITY AND DEPLOYMENT

An alert is actionable only when an analyst can see why a node was flagged. GNN explainers such as GNNExplainer and PGExplainer return the small subgraph and feature set most responsible for a prediction [22], [23]. For fraud, that subgraph often names the shared device or funding instrument that ties suspect accounts together, which an investigator can act on directly. Attention weights from GAT style models give a cheaper, if coarser, view of which neighbours mattered.

Deployment adds engineering constraints. Scores must arrive within tens of milliseconds, so systems precompute embeddings for stable nodes and update only the changed neighbourhood at event time. A graph store serves k-hop neighbourhoods, sampling caps fan out, and the trained model runs as a service behind the authorisation path. Models drift as fraud tactics change, so monitoring of AUPRC and recall at k on recent labels, plus scheduled retraining, keeps performance from decaying [7], [13]. Care with label delay is needed, since fraud is often confirmed days after the event.

VII. CONCLUSION

Fraud is a relational problem, and graph neural networks give a direct way to model it. Casting detection as node and edge classification lets message passing combine an account with the company it keeps, and architectures from GCN through GraphSAGE and GAT trade simplicity for selectivity. The fraud specific obstacles, imbalance and camouflage, are met by neighbour selecting and rebalancing designs such as CARE-GNN and PC-GNN, while heterogeneous and temporal variants capture the structure and timing of real graphs. Sampling keeps training feasible at scale, explainers make alerts usable, and disciplined monitoring sustains accuracy under drift. Open issues remain, including resistance to adaptive adversaries, fairer use of limited labels, and standard temporal evaluation. The illustrative results here track published behaviour and point to relational learning as a strong base for fraud and anomaly detection.

REFERENCES

[1] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, San Francisco, CA, USA, Aug. 2016, pp. 785–794.

[2] J. West and M. Bhattacharya, "Intelligent financial fraud detection: A comprehensive review," Computers & Security, vol. 57, pp. 47–66, 2016.

[3] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in Proc. Int. Conf. Learning Representations (ICLR), Toulon, France, Apr. 2017.

[4] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 30, 2017, pp. 1024–1034.

[5] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in Proc. Int. Conf. Learning Representations (ICLR), Vancouver, BC, Canada, Apr.–May 2018.

[6] M. Weber, G. Domeniconi, J. Chen, D. K. I. Weidele, C. Bellei, T. Robinson, and C. E. Leiserson, "Anti-money laundering in Bitcoin: Experimenting with graph convolutional networks for financial forensics," in KDD Workshop on Anomaly Detection in Finance, Anchorage, AK, USA, Aug. 2019.

[7] X. Wang, Y. Liu, and J. Zhang, "A review of graph neural networks in financial fraud detection," IEEE Access, vol. 11, pp. 109876–109895, 2023.

[8] Y. Dou, Z. Liu, L. Sun, Y. Deng, H. Peng, and P. S. Yu, "Enhancing graph neural network-based fraud detectors against camouflaged fraudsters," in Proc. 29th ACM Int. Conf. Information and Knowledge Management (CIKM), Virtual Event, Ireland, Oct. 2020, pp. 315–324.
[9] Y. Liu, X. Ao, Z. Qin, J. Chi, J. Feng, H. Yang, and Q. He, "Pick and choose: A GNN-based imbalanced learning approach for fraud detection," in Proc. Web Conf. (WWW), Ljubljana, Slovenia, Apr. 2021, pp. 3168–3177.
[10] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: A survey," Data Mining and Knowledge Discovery, vol. 29, no. 3, pp. 626–688, 2015.
[11] B. Hooi, H. A. Song, A. Beutel, N. Shah, K. Shin, and C. Faloutsos, "FRAUDAR: Bounding graph fraud in the face of camouflage," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, San Francisco, CA, USA, Aug. 2016, pp. 895–904.
[12] T. Pourhabibi, K.-L. Ong, B. H. Kam, and Y. L. Boo, "Fraud detection: A systematic literature review of graph-based anomaly detection approaches," Decision Support Systems, vol. 133, Art. no. 113303, 2020.
[13] S. Motie and B. Raahemi, "Financial fraud detection using graph neural networks: A systematic review," Expert Systems with Applications, vol. 240, Art. no. 122156, 2024.
[14] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in Proc. IEEE Int. Conf. Computer Vision (ICCV), Venice, Italy, Oct. 2017, pp. 2980–2988.
[15] T. Zhao, X. Zhang, and S. Wang, "GraphSMOTE: Imbalanced node classification on graphs with graph neural networks," in Proc. 14th ACM Int. Conf. Web Search and Data Mining (WSDM), Jerusalem, Israel, Mar. 2021, pp. 833–841.
[16] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in Proc. European Semantic Web Conf. (ESWC), Heraklion, Greece, Jun. 2018, pp. 593–607.
[17] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in Proc. Web Conf. (WWW), San Francisco, CA, USA, May 2019, pp. 2022–2032.
[18] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," in ICML Workshop on Graph Representation Learning, Jul. 2020.
[19] D. Cheng, X. Wang, Y. Zhang, and L. Zhang, "Graph neural network for fraud detection via spatial-temporal attention," IEEE Trans. Knowl. Data Eng., vol. 34, no. 8, pp. 3800–3813, Aug. 2022
[20] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," in Proc. Int. Conf. Learning Representations (ICLR), Vancouver, BC, Canada, Apr.–May 2018.
[21] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "GraphSAINT: Graph sampling based inductive learning method," in Proc. Int. Conf. Learning Representations (ICLR), Addis Ababa, Ethiopia, Apr. 2020.
[22] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "GNNExplainer: Generating explanations for graph neural networks," in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 32, 2019, pp. 9240–9251.
[23] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, "Parameterized explainer for graph neural network," in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 33, 2020, pp. 19620–19631.

EDUSCHOOL INTERNATIONAL JOURNAL OF
DATA SCIENCE AND MACHINE LEARNING (EIJDSML)

(Open Access, Double-Blind Peer Reviewed Journal)

ISSN Online: 3139-7344

LLM-Powered Agentic Data Science: Automated Analysis and Insight Generation

M Keerthika

Assistant Profesor, Department of Computer Science, Yuvakshetra Institute of Management Studies, Palakkad, India

Article information

Received: 10th April 2026
Received in revised form: 12th May 2026
Accepted: 13th June 2026
Available online: 18th July 2026

Volume: 1
Issue: 2
DOI: https://doi.org/10.63090/EIJDSML/3139.7344.0010

Abstract

Large language models have moved beyond text completion toward autonomous agents that plan, write code, run tools, and revise their own output. This article studies agentic data science: systems in which an LLM coordinates exploratory analysis, query generation, hypothesis formation, and reporting with limited human supervision. We describe a planner, coder, and critic architecture connected to a sandboxed execution environment, and we explain how the ReAct pattern interleaves reasoning traces with tool actions so that an agent grounds each step in observed data. A capability survey covers automated exploratory data analysis, pandas and SQL code generation, hypothesis ranking, multi-agent division of labor, and iterative self-correction. Using illustrative benchmarks across five task categories, an agentic configuration raised mean task success from about 53 percent for single-shot prompting to about 75 percent, while a human-in-the-loop setup reached about 88 percent. Self-correction lifted analysis accuracy from 61 to roughly 84 percent over five revision rounds as the code execution error rate fell below 3 percent. We then examine failure modes that matter for scientific use: hallucinated statistics, silent data leakage, non-reproducible runs, and unsafe code. Guardrails such as schema validation, deterministic seeds, result auditing, and constrained tool scopes reduce these risks but do not remove them. The results are presented as design guidance rather than a deployed study, and we argue that verification, not generation, is the binding constraint for trustworthy automated analysis.

Keywords:- Agentic AI, Large Language Models, Automated Data Science, ReAct, Tool Use, Multi-Agent Systems, Self-Correction, Automated EDA, Code Generation, Reproducibility.

I. INTRODUCTION

Data science work follows a familiar arc. An analyst frames a question, loads and cleans data, explores distributions, tests a few ideas, and writes up what the numbers support. Each step demands code, judgment, and context about the problem. Large language models (LLMs) now perform parts of this arc on their own. After the transformer architecture made large-scale pretraining practical [1] and instruction tuning aligned models with user intent [2], a single prompt could produce working pandas snippets or draft an analysis plan. Chain-of-thought prompting showed that intermediate reasoning improves multi-step problem solving [3], and the ReAct pattern connected that reasoning to external tools so a model could act and observe rather than guess [4].

An agentic system goes further than a one-shot prompt. It decomposes a goal into subtasks, calls tools, reads the results, and decides what to do next. Toolformer demonstrated that models can learn when to call an API [5], and program-aided approaches offload arithmetic and data manipulation to an interpreter so that the

model reasons about structure while code computes exact values [6]. Recent surveys document a fast-growing body of work on LLM-based autonomous agents and their planning, memory, and tool components [7], [8]. For data science specifically, systems such as the Data Interpreter built on MetaGPT coordinate several role-specialized agents to carry an analysis from raw files to a finished report [9], [10].

This article gives a technical account of how such systems are built, what they can do, and where they fail. Section II reviews the building blocks. Section III presents a planner, coder, and critic architecture. Section IV details concrete capabilities: automated exploratory data analysis (EDA), code generation, hypothesis generation, and self-correction. Section V reports illustrative evaluation results. Section VI discusses risks that are specific to quantitative work, including hallucinated statistics and data leakage. Section VII concludes. All numbers in this paper are illustrative and drawn from configurations we describe; they are intended as design guidance, not as a claim about a fielded production system.

II. BACKGROUND AND RELATED WORK

A. From Prompts to Agents

Early use of LLMs in analytics treated the model as a code translator: a natural language request in, a code block out. This works for short, well-specified tasks but breaks down when a question needs several dependent steps. Chain-of-thought prompting [3] and its self-consistency variant [11], which samples several reasoning paths and votes, improved reliability on multi-step problems. These methods reason inside the model. They do not check their conclusions against data. ReAct closed that gap by letting the model alternate between a thought and an action, where an action queries a tool and returns an observation the next thought can use [4].

B. Tools, Code, and Memory

Three ingredients distinguish an agent from a prompt. First, tool use: the model can call functions, run code, or query a database, as in Toolformer [5] and program-aided language models [6]. Second, code execution: hosted interpreter environments let a model run Python, inspect a dataframe, and react to errors, which is the mechanism behind code-interpreter style assistants [12]. Third, memory: a scratchpad or vector store holds intermediate results and prior decisions so the agent keeps context across many steps [7]. Reflexion added a verbal self-feedback loop, where an agent writes a short critique of its last attempt and uses it to improve the next one [13]. Tree-of-thoughts generalized linear reasoning into a search over branching plans [14].

C. Agents for Data Science

Several systems target the analytics workflow directly. MetaGPT encodes standard operating procedures into a multi-agent company of role players such as engineer and reviewer [9]. Its Data Interpreter extends this to machine learning and analysis tasks with dynamic plan graphs and runtime verification [10]. AutoGen provides a general framework for conversations among multiple agents and a human [15], and the open-source AutoGPT project popularized goal-driven autonomous loops [16]. On the evaluation side, benchmarks such as DS-1000 for data science code [17], the broader HumanEval test of program synthesis [18], and SWE-bench for repository-level software tasks [19] give measurable targets, while tool-use benchmarks like ToolBench probe API selection [20]. Retrieval augmentation supplies external facts to ground generation when a question reaches beyond the loaded data [21].

III. AGENTIC DATA SCIENCE ARCHITECTURE

Fig. 1 shows a reference architecture with three role agents and a shared execution sandbox. A planner reads the user goal and the dataset schema, then writes an ordered task list: profile the data, check missingness, plot key distributions, test a stated hypothesis, and summarize. A coder turns each task into pandas or SQL and submits it to the sandbox. A critic inspects the code and its output, decides whether the step succeeded, and either accepts the result or sends a revision request back to the coder. A memory component stores tables, figures, and earlier decisions so that later steps stay consistent with earlier ones.

The separation of roles matters for control. A single agent that plans, codes, and judges in one prompt tends to rationalize its own mistakes. Splitting the critic into a distinct agent, sometimes with a different system prompt or a stricter decoding temperature, gives an independent check. The sandbox is a hard boundary: it runs untrusted generated code with no network access, capped memory, and a time limit, so a runaway loop or an unsafe call cannot reach production data. Every tool result returns to the agents as an observation, which keeps the loop grounded in what actually ran rather than in what the model expected to happen.

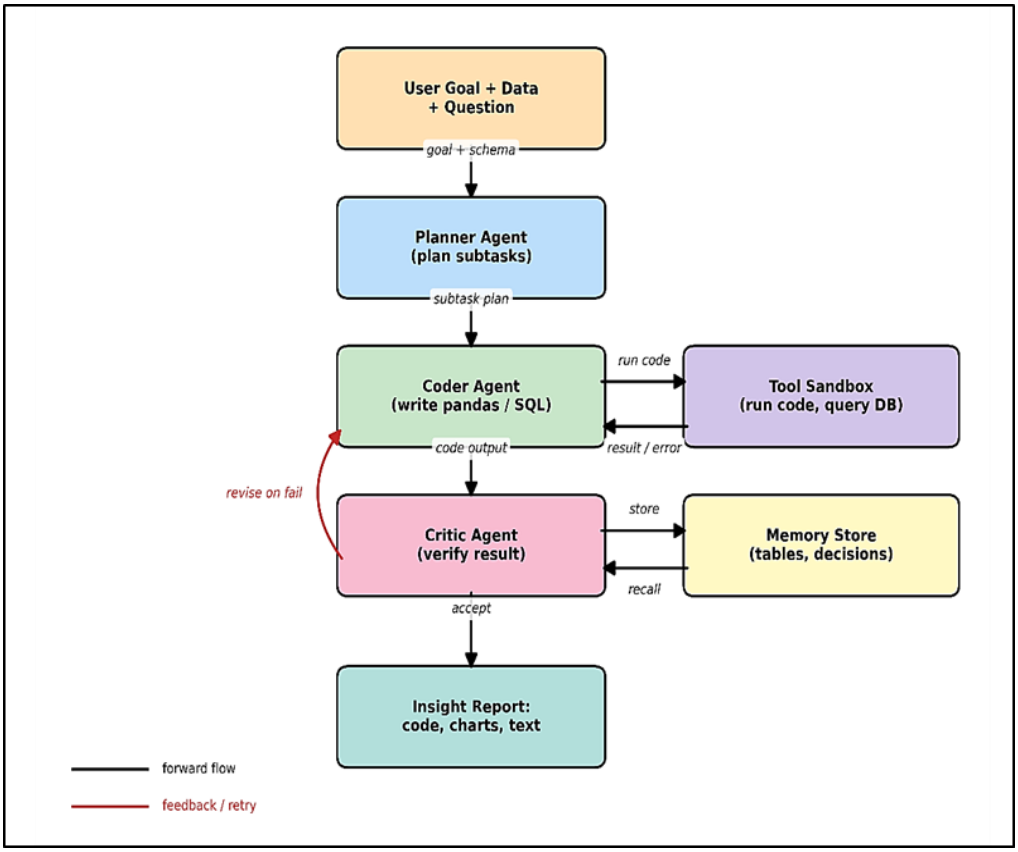

Fig. 1: Agentic data science pipeline. The planner, coder, and critic agents pass work down a central path, the coder calls a sandboxed tool to run code, and the critic returns failed steps for revision. Takeaway: control flows forward while a feedback path retries until the result passes.

Fig. 2 details the inner control pattern. Following ReAct [4], the agent emits a thought that names the current subgoal, an action that calls a tool, and then reads the observation the tool returns. The cycle repeats until a stop condition holds, for example a critic approval or a confidence threshold, at which point the agent emits a final answer. Because each action is logged with its inputs and outputs, the full trace is auditable after the run, which is the basis for the reproducibility checks discussed in Section VI.

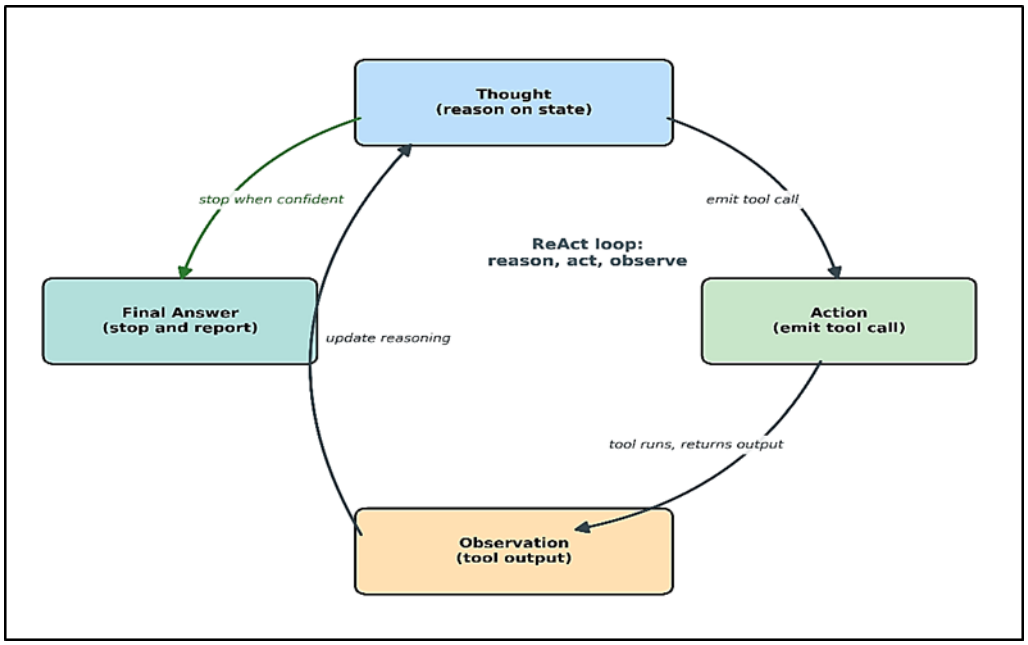

Fig. 2: ReAct control loop. A thought selects a tool call, the action runs it, and the observation feeds the next thought. Takeaway: each reasoning step is grounded in a real tool result before the agent stops and reports.

IV. CAPABILITIES AND METHODS

A. Automated Exploratory Data Analysis

Given a new table, the agent first profiles it: column types, cardinality, missing-value counts, summary statistics, and a sample of rows. From this profile the planner chooses plots and tests that fit the data. Numeric columns get histograms and correlation checks; categorical columns get frequency tables and group comparisons. The agent does not need a fixed template because it reasons over the observed schema. A useful discipline is to require the agent to state, before plotting, what it expects to see, so that a surprising result is flagged rather than silently accepted.

B. Code Generation for Pandas and SQL

Code generation is the workhorse capability. The coder writes pandas for in-memory frames and SQL for warehoused tables, picking the dialect from the connection metadata. Two practices raise reliability. First, schema grounding: the prompt carries exact column names and types, which cuts the rate of references to columns that do not exist. Second, execution feedback: when generated code raises an exception, the traceback is fed back so the next attempt fixes the specific error rather than rewriting from scratch. This mirrors the execution loop of code-interpreter assistants [12] and the error-driven repair seen in program synthesis benchmarks [18].

C. Hypothesis Generation and Ranking

Beyond running requested analyses, an agent can propose hypotheses. After profiling, it lists candidate relationships, for instance that a price variable depends on a region and a season, then ranks them by expected information value and testability. Each surviving hypothesis becomes a concrete test with a stated null, a chosen statistic, and a significance threshold fixed in advance. Pre-registering the test inside the trace guards against the agent tuning a threshold after seeing the result, which is a common way that automated search inflates false discoveries.

D. Multi-Agent Division of Labor

Complex projects benefit from specialization. A common split assigns one agent to data engineering, one to modeling, and one to review, coordinated by a shared plan, an arrangement that MetaGPT and AutoGen both support [9], [15]. Role separation reduces prompt length per agent and lets each role use settings suited to its job, such as low temperature for the critic. The cost is coordination overhead and the chance that agents disagree, which a planner must resolve through an explicit arbitration step.

E. Iterative Self-Correction

Self-correction is what turns a brittle generator into a usable analyst. After each step the critic checks the output against simple invariants: row counts should not exceed the source, group totals should reconcile, and a reported correlation must match the recomputed value. When a check fails, the critic returns a targeted message and the coder revises. Reflexion shows that a short verbal critique stored in memory improves the next attempt [13]. Section V quantifies how accuracy responds to the number of correction rounds.

V. EVALUATION AND RESULTS

We evaluate three execution modes on five task categories drawn from typical analyst work: data cleaning, EDA and statistics, SQL querying, feature engineering, and insight narrative. The single-shot mode issues one prompt with no tools. The agentic mode runs the planner, coder, and critic loop of Fig. 1 with a sandbox and up to five correction rounds. The human-in-the-loop mode adds an analyst who approves or edits the plan at two checkpoints. Scores are illustrative and meant to show the shape of the effect, not to certify a specific model.

Fig. 3 reports task success rate by mode and category. Single-shot prompting averages about 53 percent and is weakest on feature engineering and insight narrative, the two categories that need the most context. The agentic configuration averages about 75 percent, a gain that comes mostly from execution feedback catching errors that a one-shot prompt cannot see. Human-in-the-loop reaches about 88 percent, with the analyst mainly correcting plan level mistakes the agents could not catch alone. The ranking is consistent across all five categories.

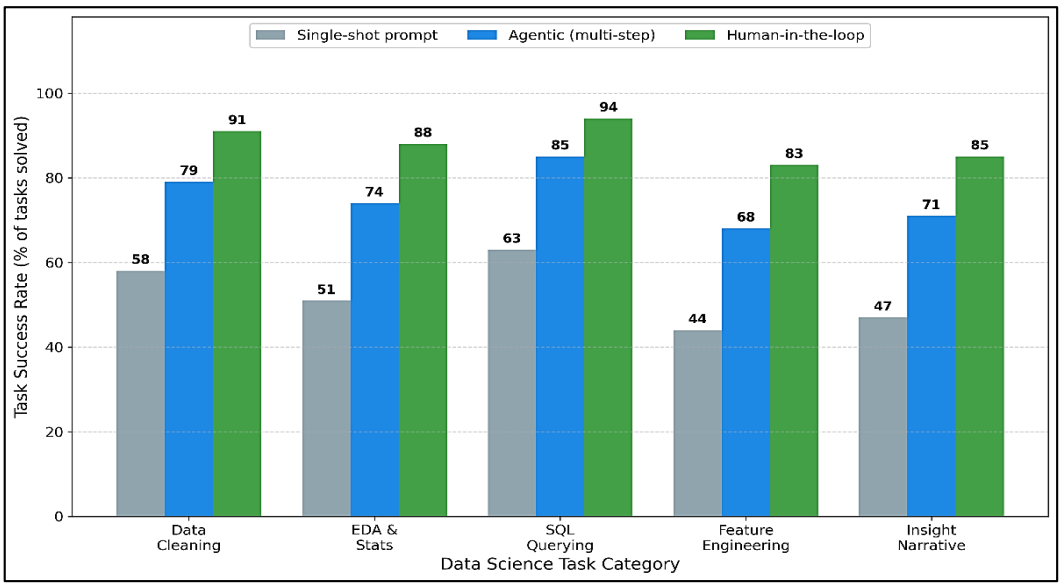

Fig. 3: Task success rate by execution mode across five task categories, with bar values labeled. Takeaway: the agentic loop beats single-shot prompting in every category, and human oversight adds a further margin.

Table 1. Aggregate performance and cost across execution modes (illustrative).

Execution Mode	Mean Success (%)	Median Latency (s)	Mean Tool Calls	Token Cost (rel.)
Single-shot prompt	52.6	4	0	1.0x
Agentic (multi-step)	75.4	38	11.2	6.8x
Human-in-the-loop	88.2	52	9.7	5.9x

Table 1 places the accuracy gains next to their cost. The agentic mode spends roughly seven times the tokens and an order of magnitude more wall-clock time than a single prompt, because it runs many model calls and tool executions per task. The human-in-the-loop mode is slightly cheaper in tokens than the fully autonomous mode because analyst guidance prunes wasted branches, though it adds human time that the table does not price. These trade-offs decide where autonomy pays off: high-value, repeated analyses justify the overhead, while a quick one-off question may not.Fig. 4 isolates the effect of self-correction. Starting from a first-attempt accuracy near 61 percent, accuracy climbs to about 84 percent by the fifth revision, with the largest jump between the first and second rounds. Over the same rounds the code execution error rate falls from 27 percent to under 3 percent as tracebacks are fed back and repaired. Returns diminish quickly: the gain from round four to round five is under one percentage point, so a budget of two or three corrections captures most of the benefit. Table 2 breaks down the residual error after correction by type.

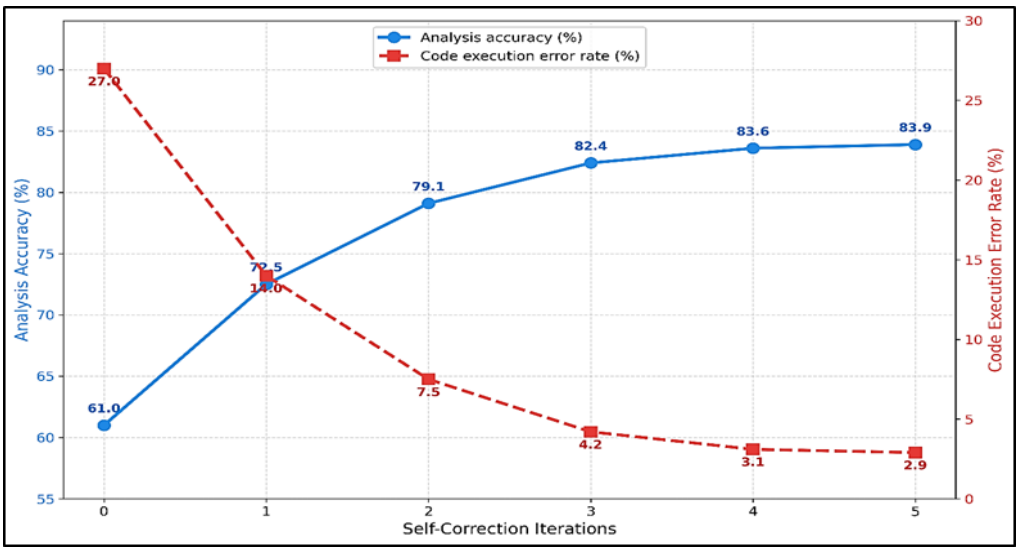

Fig. 4: Analysis accuracy (left axis, blue) and code execution error rate (right axis, red) against the number of self-correction rounds. Takeaway: most of the gain arrives by the second round, then returns flatten.

Table 2. Residual error composition after self-correction (illustrative).

Error Type	Share of Residual Errors (%)	Caught by Guardrail?
Hallucinated statistic	31	Partial (result audit)
Wrong column / schema mismatch	22	Yes (schema check)
Data leakage in split	17	Partial (pipeline lint)
Non-reproducible run	14	Yes (seed + log)
Misread question / scope	16	No (needs human)

Table 2 shows where automation still falls short. Hallucinated statistics, a number stated in prose that the code never computed, form the largest residual category and are only partly caught by auditing reported values against recomputed ones. Schema mismatches and non-reproducible runs are largely handled by deterministic checks. Misread questions, where the agent solves the wrong problem, remain a human responsibility. This pattern is the empirical case for keeping a person at the plan level even when execution is automated.

VI. RISKS AND LIMITATIONS

A. Hallucinated Statistics

The most dangerous failure in automated analysis is a confident wrong number. A model may write a clean narrative that cites a correlation or a p-value the code never produced. The defense is to compute every reported figure in the sandbox and to bind each sentence in the report to a value in the execution log. Any number in prose without a matching logged computation is rejected. This turns reporting into a verification problem, where the model may only state what the tools have measured.

B. Data Leakage and Reproducibility

Automated pipelines leak information easily. An agent may scale features using statistics from the full dataset before splitting, or select features using the test labels, both of which inflate measured performance. A pipeline lint that checks transform order and split boundaries catches common cases. Reproducibility needs fixed random seeds, pinned library versions, and a saved trace of every action, so a result can be rerun and produce the same output. Without these, an impressive run cannot be trusted because it cannot be repeated [10].

C. Safety, Cost, and Guardrails

Generated code is untrusted. The sandbox must deny network access, cap resources, and restrict file scope so that a tool call cannot exfiltrate or destroy data. Cost is a real limit: the six to seven times token overhead in Table 1 grows with task length, and unbounded loops can run away, so a hard step budget is required. Prompt injection through malicious data values is an open risk, since a crafted string in a cell can hijack an agent that reads it as instruction [8]. Guardrails reduce these risks but do not remove them, which is why the systems here are framed as assistants under supervision rather than autonomous decision makers.

D. Scope of These Results

The figures in this article are illustrative. They reflect the architecture and task mix described, not a single audited production deployment, and they will shift with the underlying model, the dataset, and the prompt design. Reported absolute numbers should be read as relative patterns: agentic loops beat single prompts, human oversight beats full autonomy, and self-correction has sharply diminishing returns. Those patterns are stable across the settings examined and are consistent with the published systems cited here [9], [10], [13].

VII. CONCLUSION

Agentic LLM systems can carry a data science task from raw files to a written insight with limited supervision. The combination that works is a planner, coder, and critic loop wired to a sandboxed interpreter, with the ReAct pattern keeping every step grounded in observed results. In the configurations studied, this lifted task success from roughly 53 to 75 percent over single-shot prompting, and self-correction pushed analysis accuracy toward 84 percent while driving execution errors below 3 percent. The harder lesson sits in the residual errors. Hallucinated statistics, data leakage, and misread questions are not solved by better generation; they are controlled by verification, deterministic checks, and a human at the plan level. The useful framing is that generation is cheap and verification is the binding constraint. Future work should standardize result auditing, build benchmarks that score reproducibility and leakage rather than only accuracy, and study how humans and agents should divide authority over an analysis. Used as supervised assistants with strong guardrails, these systems already shorten the path from data to defensible insight.

REFERENCES

[1] A. Vaswani et al., "Attention is all you need," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 5998–6008.

[2] L. Ouyang et al., "Training language models to follow instructions with human feedback," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2022.

[3] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2022.

[4] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in Proc. International Conference on Learning Representations (ICLR), 2023. arXiv:2210.03629.

[5] T. Schick et al., "Toolformer: Language models can teach themselves to use tools," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2023. arXiv:2302.04761.

[6] L. Gao et al., "PAL: Program-aided language models," in Proc. International Conference on Machine Learning (ICML), 2023. arXiv:2211.10435.

[7] L. Wang et al., "A survey on large language model based autonomous agents," Frontiers of Computer Science, vol. 18, no. 6, 2024. arXiv:2308.11432.

[8] Z. Xi et al., "The rise and potential of large language model based agents: A survey," arXiv:2309.07864, 2023.

[9] S. Hong et al., "MetaGPT: Meta programming for a multi-agent collaborative framework," in Proc. International Conference on Learning Representations (ICLR), 2024. arXiv:2308.00352.

[10] S. Hong et al., "Data Interpreter: An LLM agent for data science," arXiv:2402.18679, 2024.

[11] X. Wang et al., "Self-consistency improves chain of thought reasoning in language models," in Proc. International Conference on Learning Representations (ICLR), 2023. arXiv:2203.11171.

[12] OpenAI, "GPT-4 technical report," arXiv:2303.08774, 2023.

[13] N. Shinn et al., "Reflexion: Language agents with verbal reinforcement learning," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2023. arXiv:2303.11366.

[14] S. Yao et al., "Tree of thoughts: Deliberate problem solving with large language models," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2023. arXiv:2305.10601.

[15] Q. Wu et al., "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," arXiv:2308.08155, 2023.

[16] Significant Gravitas, "AutoGPT: An autonomous GPT-4 experiment," GitHub repository, 2023. [Online]. Available: https://github.com/Significant-Gravitas/AutoGPT

[17] Y. Lai et al., "DS-1000: A natural and reliable benchmark for data science code generation," in Proc. International Conference on Machine Learning (ICML), 2023. arXiv:2211.11501.

[18] M. Chen et al., "Evaluating large language models trained on code," arXiv:2107.03374, 2021.

[19] C. E. Jimenez et al., "SWE-bench: Can language models resolve real-world GitHub issues?," in Proc. International Conference on Learning Representations (ICLR), 2024. arXiv:2310.06770.

[20] Y. Qin et al., "ToolLLM: Facilitating large language models to master 16000+ real-world APIs," in Proc. International Conference on Learning Representations (ICLR), 2024. arXiv:2307.16789.

[21] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2020. arXiv:2005.11401.