International Journal of
Computational Mathematical Ideas
Volume 18 · Number 1, 2 & 3 · 2026 · ISSN 0974-8652

Available online at www.ijcmi.com

Agentic AI Automation for Data Science in Large-Scale Projects: Intelligent Agents and Loop Engineering

Arunkumar Medisetty *

Staff Software Engineer, The Home Depot,
2161 Newmarket Pkwy SE, Marietta, GA 30067, USA

*Corresponding author: arunkumar.medisetty@yahoo.com

Article Info

Article history:
Received: 18-01-2026
Revised: 17-07-2026
Accepted: 27-07-2026
Available online: 2026

Keywords:
Agentic AI; Data science automation; AutoML; Intelligent agents; Loop engineering; Hyperparameter search; Model selection.

Abstract

Automating data science on large-scale projects has become a problem of engineering loops rather than tuning models by hand. This paper takes a design stance on that problem and presents LEAP, a loop-engineered agentic pipeline in which an orchestrator agent coordinates stage agents through three feedback loops: a fast loop that searches model configurations under a compute budget, a medium loop in which a critic verifies candidates and revises the pipeline, and a slow loop that monitors and retrains under drift. We make two parts of the design precise. We show that a blind search needs on the order of $1/p$ trials to find a configuration that is within tolerance of optimal, which motivates cheap multi-fidelity screening, and we show that the optimism of selecting the best of N configurations by validation grows like $\sqrt{2\ln N}$, which motivates a cross-validated critic. We validate the design with a fully reproducible experiment written from scratch, and we report honest results. At equal compute, agentic multi-fidelity search reaches a test AUC of 0.749 against random search's 0.738 and wins in 79 percent of runs; a critic loop improves test AUC at every search intensity and curbs the validation optimism that heavy search injects, which rises to 0.037 under the heaviest search; and the critic advantage holds as dimensionality grows. Crucially, a naive greedy agent does not beat random search, which is a strong baseline. The value of agentic automation therefore lies in how it allocates compute and how it selects, not in the promise of a cleverer model, and both can be engineered and reasoned about.

1. Introduction

The practice of data science on large-scale projects is, increasingly, a practice of automation. A modern pipeline must ingest and clean data, engineer features, search over models and their hyperparameters, validate the result, and keep it healthy in production, and the sheer number of these decisions has outgrown what a team can tune by hand [18]. Automated machine learning arose to close this gap, turning model and hyperparameter selection into a search problem that a computer can run [6]. The recent turn to agentic systems, in which a language-model agent reasons, calls tools, and acts in a loop rather than emitting a single answer, promises to extend that automation across the whole lifecycle [1].

The promise is attractive but the failure modes are specific. An automated pipeline that searches hard over a fixed validation split will eventually overfit that split and report a model that looks better than it is [13]. A search that is cheap to run in the small becomes expensive at scale, so the way compute is allocated across candidates matters as much as the search itself [11]. And a model that passes offline is only the start; most of the cost of a deployed system is incurred afterwards, in monitoring, maintenance, and adaptation to drift [17]. Automation that ignores these realities does not save work, it moves the work downstream and hides it.

This paper takes a design and engineering stance on the problem. We treat the automation of a large-scale data-science project as a set of engineered loops: a fast loop that searches configurations under a compute budget, a medium loop in which a critic verifies candidates and revises the pipeline, and a slow loop that monitors and retrains under drift. We call the resulting system LEAP, a Loop-Engineered Agentic Pipeline. Our contributions are three. First, we give a

reference architecture in which an orchestrator agent coordinates stage agents and a critic guards the pipeline against its own overfitting. Second, we make two parts of the design precise: we bound the number of trials a blind search needs to find a good configuration, which motivates cheap multi-fidelity screening, and we bound the optimism that heavy search injects into validation estimates, which motivates the critic loop. Third, we validate the design with a fully reproducible experiment, reporting honest results, including the finding that random search is a strong baseline and that the real value of the agentic loop lies in adaptive compute allocation and disciplined selection rather than in cleverer local moves.

2. Background: Automating the Data Science Lifecycle

2.1. Automated machine learning and search

Automated machine learning frames model building as optimisation over a configuration space [6]. The workhorse methods search that space: grid and random search [9], Bayesian optimisation with a surrogate model [10], combined algorithm selection and hyperparameter tuning [8] and its automated-pipeline successors [7], and, for neural models, architecture search [12]. A recurring and sobering result is that random search is a strong baseline: because good configurations often depend on only a few dimensions, sampling uniformly is hard to beat and is trivially parallel [9]. The clearest wins come not from smarter local moves but from spending compute adaptively, evaluating many candidates cheaply and promoting only the promising ones [11]. We take this lesson directly into the design.

2.2. Intelligent agents and loop-based reasoning

The agentic literature supplies the control structure. Interleaving reasoning with actions lets an agent gather evidence before it commits [1]; tool use grounds its steps in real computations rather than parametric memory [2]; self-reflection lets it learn from its own failed attempts [3]; and multi-agent conversation lets specialised agents divide a task and check one another [5]. Surveys of this area describe a common pattern of planning, acting, observing, and revising in a loop [4]. Framed this way, a data-science automation system is a multi-agent loop whose stages map onto the lifecycle and whose feedback edges implement search, verification, and adaptation.

2.3. Machine learning systems and deployment

The systems literature supplies the constraints. Deployed machine learning accumulates hidden technical debt, in glue code, pipeline jungles, and undeclared data dependencies, that dwarfs the modelling effort [17]. Case studies of real teams show that data management, testing, and monitoring dominate the engineering budget [18], and surveys of production deployments catalogue the same challenges across industries [19]. Data itself is a first-class concern, with its own lifecycle of validation, versioning, and monitoring [20], and production readiness can be scored against explicit rubrics for testing and maintainability [21]. A credible automation design must therefore close the loop past deployment, not stop at a validation number.

3. The Automation Problem and Design Principles

We can now state the problem and the principles that follow from this literature. A large-scale data-science project presents a configuration space of pipelines, a finite compute budget, and finite held-out data, and asks for a model that generalises. Three principles guide the design.

Allocate compute, do not just sample. Because random search is already a strong baseline [9], an agent earns its place by spending compute adaptively: screening many candidates cheaply and promoting only the best, as in bandit-style successive halving [11], which allocates a budget across candidates like a multi-armed bandit [24].

Verify against the search itself. The more configurations a pipeline tries, the more it risks selecting one that merely fit the validation split [13]. A critic that re-estimates candidates by cross-validation, in the spirit of a reusable holdout, restores validity [14], cross-validation being the classical estimator of this kind [15]. Feature selection is part of this same discipline, since selecting features on the same data used to evaluate them leaks information [16].

Engineer the loop past deployment. The pipeline must monitor outcomes, watch for concept drift, and retrain, because a model that was valid yesterday can silently decay [22], a problem whose detection and adaptation are by now well reviewed [23]. These principles motivate an architecture of nested loops at different timescales.

4. Reference Architecture: LEAP

Figure 1 shows LEAP. The forward path is the familiar lifecycle: data sources feed an ingest-and-clean stage, then feature engineering, then a model-search stage that proposes configurations, ending in a validated model. What makes the architecture agentic and loop-engineered are the control elements below it. An orchestrator agent coordinates

the stage agents [5]. A critic and verifier sits between search and deployment: it re-selects candidate configurations by cross-validation, checks for leakage and drift, and either accepts a model or sends the pipeline back to revise its features or search space [14]. A monitor-and-audit layer records outcomes and triggers retraining when drift is detected [23]. Three loops run at different speeds: a fast loop of configuration search, a medium loop of critic-driven pipeline revision, and a slow loop of drift-triggered retraining.

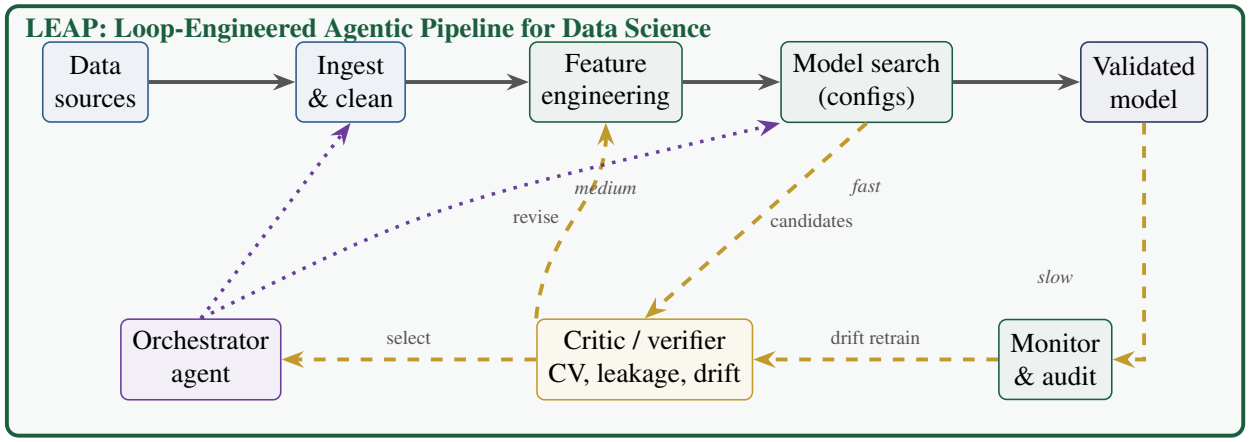

Figure 1. The LEAP reference architecture. A forward lifecycle (data to validated model) is wrapped by an orchestrator agent, a critic and verifier, and a monitoring layer. Three feedback loops run at different timescales: fast configuration search, medium critic-driven revision, and slow drift-triggered retraining.

5. Formal Model: Search and Selection

Two parts of the design admit clean guarantees. Consider a configuration space and a budget of evaluations.

Proposition 1 (Random-search hitting time)**.** *Suppose a fraction p of configurations is within ε of optimal. Under uniform random sampling, the number of draws until the first such configuration is Geometric with parameter p and expectation $1/p$, and the probability that at least one of T draws is good is $1-(1-p)^T$. To succeed with probability $1-\delta$ it suffices to take $T \geq \ln(\delta)/\ln(1-p)$ draws.*

Proof. Draws are independent and each is good with probability p, so the index of the first good draw is Geometric(p) with mean $1/p$. The probability that T independent draws are all bad is $(1-p)^T$, so the complement is $1-(1-p)^T$; setting this at least $1-\delta$ and solving gives the stated bound. □

Proposition 1 explains why blind search is expensive when good configurations are rare, that is when p is small, as happens in high dimensions. It also explains the remedy: if cheap low-fidelity evaluations let the system screen η^2 times as many candidates for the same compute, the effective number of trials T rises by the same factor, and by the proposition the probability of finding a good configuration rises with it. This is the principle behind the multi-fidelity search in our experiments [11].

Proposition 2 (Optimism of selection)**.** *Let $\hat{R}_1,\dots,\hat{R}_N$ be validation estimates of the true risks $R_1,\dots,R_N$ of N configurations, with each error $\hat{R}_i - R_i$ zero-mean and sub-Gaussian with parameter σ. Then the expected optimism of the best-by-validation configuration satisfies*

$$\mathbb{E}\left[\max_{1\leq i\leq N}(\hat{R}_i - R_i)\right] \;\leq\; \sigma\sqrt{2\ln N}. \tag{1}$$

If the validation estimate uses n_{val} points, then $\sigma = c/\sqrt{n_{\mathrm{val}}}$ for a constant c, so the optimism grows like $c\sqrt{2\ln N/n_{\mathrm{val}}}$.

Proof. For any $s>0$, by Jensen's inequality and the sub-Gaussian moment bound, $\exp(s\,\mathbb{E}[\max_i Z_i]) \leq \mathbb{E}[\max_i e^{sZ_i}] \leq \sum_i \mathbb{E}[e^{sZ_i}] \leq N e^{s^2\sigma^2/2}$, where $Z_i = \hat{R}_i - R_i$. Taking logs gives $\mathbb{E}[\max_i Z_i] \leq \ln N/s + s\sigma^2/2$; minimising over s at $s=\sqrt{2\ln N}/\sigma$ yields (1). The scaling of σ with n_{val} is the standard error of a mean of n_{val} bounded terms [25]. □

Proposition 2 is the reason a search that tries more configurations N reports an increasingly optimistic validation score, an instance of selection bias in model selection [13]. The optimism shrinks if the estimate is more stable, that is if n_{val} is effectively larger, which is exactly what a cross-validated critic achieves by averaging over folds [15]. Algorithm 2 states the LEAP loop that puts both propositions to work.

Algorithm 1: LEAP, the loop-engineered agentic pipeline

Input: data; configuration space; compute budget; fidelity factor η.

1. orchestrator runs ingest, cleaning, and feature engineering (stage agents)
2. propose a set of candidate configurations (search)
3. **fast loop:** evaluate candidates at low fidelity; promote the top $1/\eta$ to higher fidelity; repeat (successive halving)
4. **critic:** re-select survivors by K-fold cross-validation; check leakage and calibration
5. **medium loop:** if the critic rejects, revise features or search space and return to step 2
6. fit and validate the final model; record provenance to the audit log
7. **slow loop:** monitor outcomes and drift; on drift, retrain

Output: validated model with provenance.

Figure 2. The LEAP loop. Step 3 implements the multi-fidelity screening motivated by Proposition 1; step 4 implements the cross-validated critic motivated by Proposition 2; steps 6 to 7 close the oversight loop.

6. Experiments

We validate the design with a reproducible experiment written from scratch in numpy, with a fixed seed, so that every number and figure regenerates. The task is a sparse logistic-regression problem with weak signal spread over a few features among many noise features, and small training and validation sets, a regime in which both model fitting and model selection can overfit. The pipeline searches over the number of features kept and the regularisation strength; performance is measured by held-out AUC. No external learning library is used.

6.1. Multi-fidelity search versus random search

We compare an agentic multi-fidelity loop, a successive-halving schedule that screens candidates on subsampled data and promotes the best to full data, against random search at *equal* total compute. Table 1 and Figure 3 report the result: at a matched budget of nine full-training-fit units, the multi-fidelity loop selects a model with test AUC 0.749 against random search's 0.738, and it matches or beats random search in 79% of runs. The gain is real but modest, and it comes from screening roughly three times as many candidates for the same compute, exactly as Proposition 1 predicts [11]. Consistent with the literature, random search is a strong baseline that a naive local search does not beat [9].

Table 1. Multi-fidelity agentic search versus random search at equal compute. The agentic loop screens more candidates for the same budget and selects a better model.

Method	Compute (units)	Test AUC	Wins vs random
Random search	9.0	0.738	–
Agentic multi-fidelity (SH)	9.0	**0.749**	79%

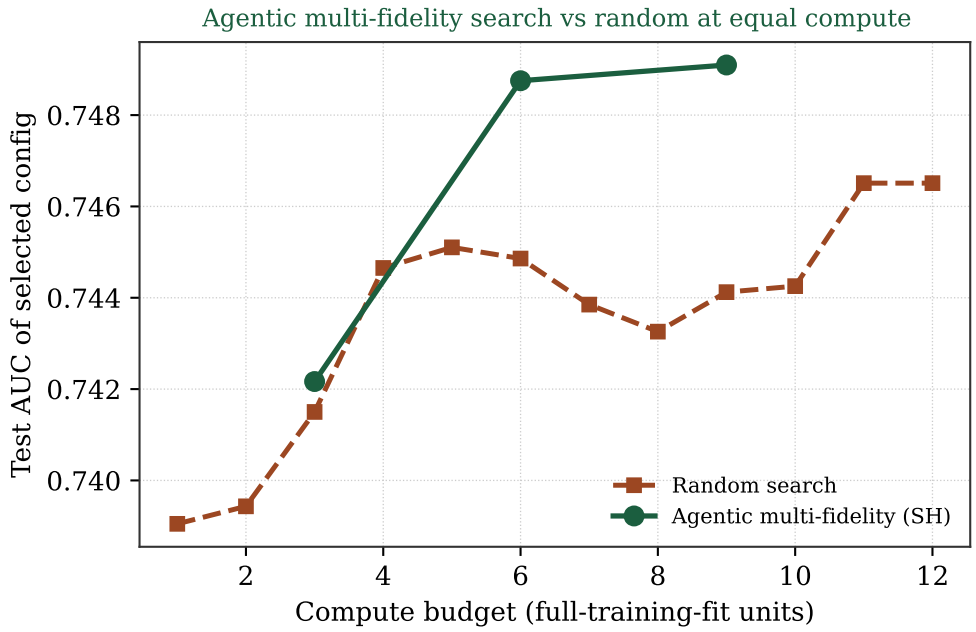

Figure 3. Test AUC of the selected configuration against compute budget. The agentic multi-fidelity loop reaches a higher test AUC than random search at equal compute.

6.2. Validation overfitting and the critic loop

Next we isolate the selection problem. A naive automation picks the configuration with the best score on a single validation split; the critic loop instead re-selects the candidates by cross-validation on pooled data. Table 2 and Figure 4 show two effects. First, the naive method's validation optimism, the gap between its selected validation score and the true test score, grows with the number of configurations searched, reaching 0.037 at $B = 180$, precisely the selection bias bounded in Proposition 2 [13]. Second, the critic loop delivers a higher test AUC at every search intensity, because cross-validation gives a more stable estimate and is harder to overfit [14].

Table 2. Validation overfitting versus the critic loop. Naive selection becomes more optimistic as more configurations are tried; the critic loop generalises better.

Configs B	Optimism (naive)	Test AUC (naive)	Test AUC (critic)
5	0.012	0.742	**0.769**
40	−0.001	0.765	**0.771**
90	0.010	0.746	**0.760**
180	0.037	0.768	**0.770**

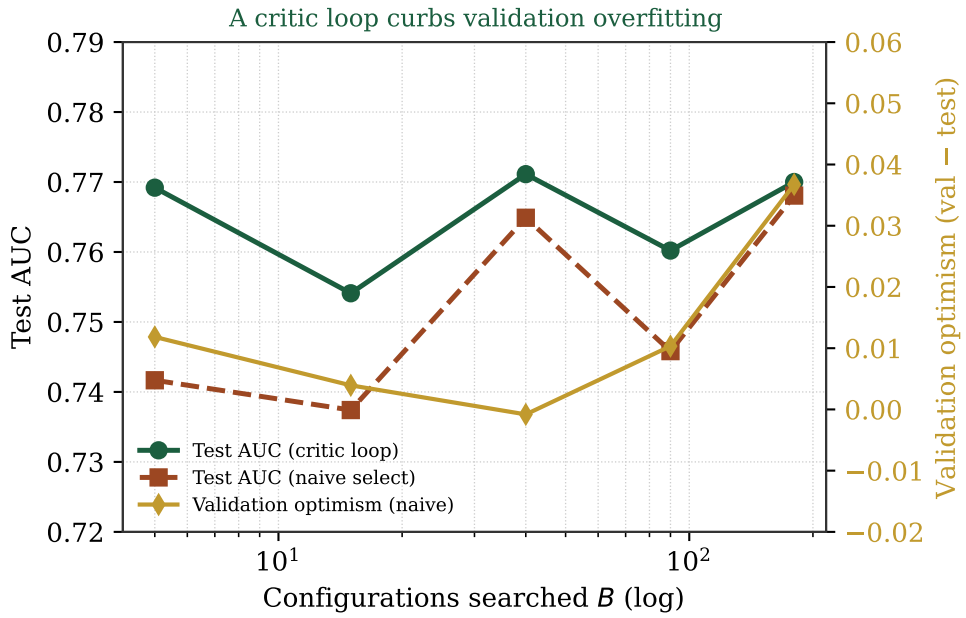

Figure 4. The critic loop against validation overfitting. As the number of searched configurations grows, naive selection becomes optimistic (gold), while the cross-validated critic keeps test AUC higher than naive selection at every budget.

6.3. Scaling with dimensionality

Finally we vary the feature dimensionality, the setting that matters for large-scale projects with many candidate signals. Figure 5 shows that as dimensionality grows from twenty to one hundred sixty features, both methods degrade as noise features accumulate, but the critic-based loop stays ahead of naive selection throughout, from 0.793 against 0.785 at $d = 20$ to 0.744 against 0.729 at $d = 160$. The disciplined selection that the critic enforces is exactly what protects generalisation as the search space grows [16].

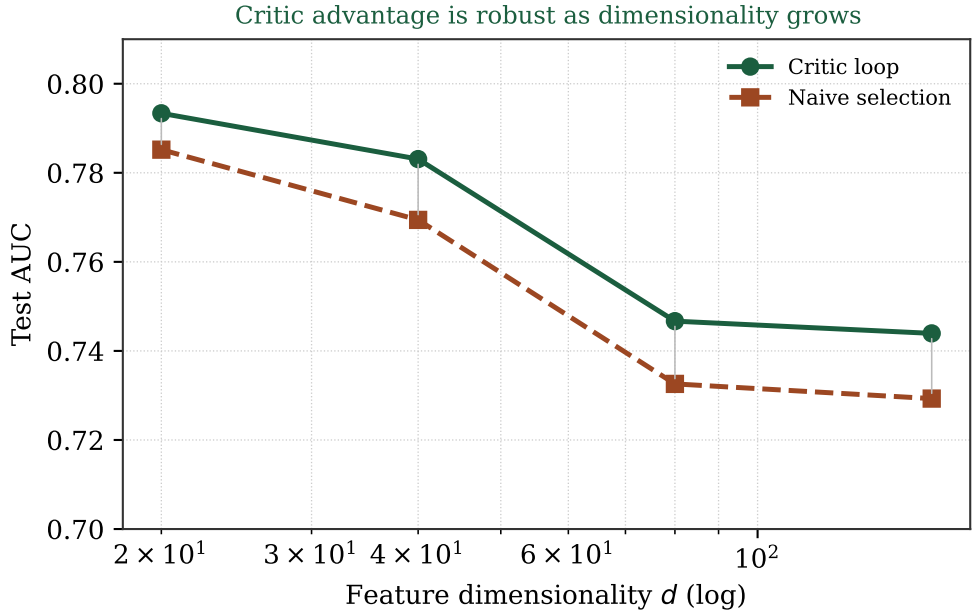

Figure 5. Scaling with dimensionality. Both methods lose accuracy as noise features accumulate, but the critic-based loop remains above naive selection at every dimensionality.

7. Discussion

The experiments support a specific and honest reading of what agentic automation buys. It does not buy a magic search that dominates random sampling; random search is a strong baseline and a naive greedy agent does not beat it [9]. What it buys is, first, better use of a compute budget through adaptive multi-fidelity screening [11], and second, disciplined selection through a critic that resists the validation overfitting which heavy search inevitably invites [13]. Both are properties of the loop, not of any single model, which is the sense in which the automation is loop-engineered.

The limitations are real. The experiment is a controlled simulation, not a production pipeline, and its data, feature engineering, and model class are deliberately simple so that every number is reproducible; real pipelines add data-quality problems, richer models, and human stakeholders that the simulation does not capture [18]. The critic assumes that cross-validation is affordable, which is not always true at scale, and that the held-out data is representative, which drift can break [22]. The architecture leans on an orchestrator and stage agents whose own reliability, cost, and failure modes we have not modelled [5]. And the accounting here is offline; the dominant costs of a deployed system are the downstream ones of monitoring and maintenance [17], a pattern documented across many deployments [19].

Future work runs in three directions: replacing the linear model and hand-built search with a tool-using language-model agent inside the same loops [2], one that reflects on its own failed attempts [3], learning the fidelity schedule and the critic's acceptance rule online [24], and extending the audit and drift monitor into a continuous data-and-model observability layer for production [20], with explicit production-readiness testing [21] and continuous drift tracking [23].

8. Conclusion

We treated the automation of large-scale data science as a problem of loop engineering and gave LEAP, a loop-engineered agentic pipeline in which an orchestrator and stage agents run a fast search loop, a medium critic-and-revise loop, and a slow monitor-and-retrain loop. We proved two guarantees that shape the design: a blind search needs on the order of $1/p$ trials to find a good configuration, which motivates cheap multi-fidelity screening, and the optimism of selecting the best of N configurations grows like $\sqrt{2\ln N}$, which motivates a cross-validated critic. A fully reproducible experiment confirmed the design and, just as importantly, reported honestly where it does not help: multi-fidelity screening beat random search at equal compute and won in most runs, the critic loop improved test performance at every search intensity and curbed validation optimism, and its advantage held as dimensionality grew, while a naive greedy agent did not beat random search at all. The lesson for large-scale projects is that the value of agentic automation lies in how it allocates compute and how it selects, not in the promise of a cleverer model, and that both can be engineered and reasoned about [4].

Acknowledgements

The author thanks colleagues for helpful discussion. All code and data needed to reproduce every figure and table are available from the author.

References

[1] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: synergizing reasoning and acting in language models," in *Proc. International Conference on Learning Representations (ICLR)*, 2023.

[2] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: language models can teach themselves to use tools," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 68539–68551.

[3] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: language agents with verbal reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 8634–8652.

[4] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, et al., "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, art. 186345, 2024.

[5] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, "AutoGen: enabling next-gen LLM applications via multi-agent conversation," arXiv:2308.08155, 2023.

[6] F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., *Automated Machine Learning: Methods, Systems, Challenges*. Cham, Switzerland: Springer, 2019.

[7] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems*, vol. 28, 2015, pp. 2962–2970.

[8] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms," in *Proc. 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2013, pp. 847–855.

[9] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[10] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, vol. 25, 2012, pp. 2951–2959.

[11] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: a novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018.

[12] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: a survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

[13] G. C. Cawley and N. L. C. Talbot, "On over-fitting in model selection and subsequent selection bias in performance evaluation," *Journal of Machine Learning Research*, vol. 11, pp. 2079–2107, 2010.

[14] C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, and A. Roth, "The reusable holdout: preserving validity in adaptive data analysis," *Science*, vol. 349, no. 6248, pp. 636–638, 2015.

[15] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. 14th International Joint Conference on Artificial Intelligence (IJCAI)*, 1995, pp. 1137–1143.

[16] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[17] 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 *Advances in Neural Information Processing Systems*, vol. 28, 2015, pp. 2503–2511.

[18] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: a case study," in *Proc. IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019, pp. 291–300.

[19] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: a survey of case studies," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–29, 2022.

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

[21] 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 International Conference on Big Data*, 2017, pp. 1123–1132.

[22] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, art. 44, 2014.

[23] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: a review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2019.

[24] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *Foundations and Trends in Machine Learning*, vol. 5, no. 1, pp. 1–122, 2012.

[25] S. Boucheron, G. Lugosi, and P. Massart, *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford, U.K.: Oxford University Press, 2013.