International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed, a Bimonthly Journal|

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802015

Self-Optimizing Pipelines ML Systems That Tune Themselves in Production

Sandeep Sarngadharan

Software Architect, SPRI Partners LLC, USA

ABSTRACT: Production machine learning pipelines are typically built, trained, and tested using historical data, then deployed with a fixed set of settings. However, real-world environments are constantly changing. Customer behavior evolves, data patterns shift, and system resources fluctuate. When performance drops, data scientists must manually investigate the problem, retune the model, and redeploy — a process that is slow, expensive, and prone to human error. This article introduces Self-Optimizing Pipelines (SOPs) — machine learning systems that automatically adjust their own configuration while running in production, without any human intervention. An SOP continuously monitors its own performance, detects when the data distribution has changed, searches for better settings (such as hyperparameters, feature selections, and ensemble weights), and safely deploys those improvements. We present a complete system architecture with five interconnected modules. Using a real-world credit card fraud detection dataset with simulated concept drift, we show that an SOP achieves an average F1-score of 0.93 compared to 0.71 for a static pipeline — a 31% improvement. The SOP reduces the need for manual data scientist intervention by 94% and adds only a small increase in processing time (from 12 milliseconds to 16 milliseconds per prediction), which is acceptable for most real-time applications. Our results demonstrate that self-tuning production systems are not only feasible but also stable, maintainable, and ready for deployment with existing open-source tools.

KEYWORDS: Self-optimizing pipelines, production machine learning, online learning, concept drift, Bayesian optimization, MLOps, multi-armed bandits, automated machine learning, adaptive systems.

I. INTRODUCTION

1.1 The Problem with Static Machine Learning Pipelines

Deployment of machine learning models has typically followed a fairly standard lifecycle. Data scientists collect historical (training) data and build (train) a model out-of-band (offline) and validate it using an independent data set (test set) before finally deploying it to a production environment. Once the model has been deployed, it stays in production until there is a manual decision to re-deploy the model with an update. As this procedure is a 'static deployment', it will not typically meet the performance needs of a changing (dynamic) production environment [11].

An example of this static nature can be seen in changes to the relationship between the model input features and the target variable, commonly referred to as concept drift. As new techniques are introduced by people committing fraud that result in them circumventing detection systems, a fraud detection model developed from only the historical transactional data will become less effective in its ability to detect fraudulent activity as that activity has continued to evolve over time.

Another example of this static nature is illustrated by the phenomenon known as data drift [2]. The statistical distributions of the input features will also change/evolve over time. For example, in a standard e-commerce website, customer viewing patterns between weekday activity and weekend activity, as well as between seasonal or holiday-related periods will be dissimilar. Therefore, a model that was constructed utilizing solely weekday data will underperform when utilized during the weekend.

There are also many system-level changes that will occur or drift over time. All variables associated with running the prediction pipeline (e.g. network latency, memory usage, processing throughput) may all therefore change due to regularly varying user-load patterns, as well as possible hardware-related failures. A prediction pipeline that is optimized for low-load periods may not operate within the limits of the stated service-level agreements during peak-load periods; thereby failing to meet required service performance [10].

When a drop in model performance is observed by manual review of an automated monitoring system, the data scientist may perform a manual process of re-tuning the model. The data scientist will typically examine the dashboards to

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed, a Bimonthly Journal|

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802015

identify the cause of the fall-off in model performance, rerun offline model-validation tests against the newly collected data since it was deployed to production, and then subsequently update the hyper-parameters and re-train the model, and finally redeploy the model to production once again. All of these activities could take several hours or even days; therefore, during this time period the model continues to produce output data that may be different from expected output data than what was produced by previous versions and could result in lost revenue, user frustration, or unsafe conditions [11].

1.2 The Vision of Self-Optimizing Pipelines

The objective of self-optimizing pipelines is to create a closed loop between monitoring and reconfiguration. Instead of waiting for someone to identify an issue in the pipeline and fix it, there will be an autonomous optimization engine that continuously analyzes the pipeline's operational performance. If it detects any significant degradation, the engine will automatically search for new and better configurations and implement them while the pipeline is running live.

The primary technical challenge is to balance two competing objectives. On one hand, the system will explore, which means it will try different, new configurations that it hasn't used in the past to see if they perform better. On the other hand, the system will exploit, or continue using, good configurations to maximize throughput. If it explores too much relative to how much it exploits, it may find a configuration that doesn't work well [15]. Conversely, if it does not explore sufficiently, it will never discover configurations with better performance than the ones currently used.

Real-life constraints add an increased level of complexity to this task. The amount of computing power available for re-optimization is limited because it must also provide enough resources to complete inference. Feedback provided to the optimization engine is typically delayed and/or noisy; for example, a transaction may take 30 minutes to receive confirmation of whether it is fraudulent, and there are many instances in which no determination is made about the transaction at all. Therefore, safety is very important; new configurations cannot cause catastrophic performance failures.

1.3 Contributions of This Work

This paper makes an important contribution to the development of self-optimizing machine learning systems by defining a formal framework and architecture that consists of five main components including: monitoring & drift detection, optimization engine, performance history buffer, configuration deployer and executable machine learning pipeline. A detailed description of a practical methodology is provided including best practices for incorporating a Bayesian optimization technique [5] to search for optimal hyper-parameter values, multi-armed bandit [8] techniques for dynamically adjusting the weights of multiple classifiers via ensembling, and AutoDM via ADWIN [3] for detecting drift in incoming data. Empirical results from a fraud detection example demonstrate the self-optimizing pipelines' superiority vs static baseline systems with respect to accuracy, latency and manual effort whilst providing insights regarding failure modes and how practitioners can make effective decisions.

The layout of this paper is as follows: Section 1 introduces the major areas of research for this paper, including concept drift, hyperparameter optimisation, bandit algorithms and self-optimising systems; Section 2 describes in detail our approach for these areas, including problem definition(s), architecture diagram, algorithms and experimental set-up; Section 3 provides empirical evidence to support our approach (i.e. charts/tables); Section 4 discusses implications of our approach with respect to trade-offs or limitations associated with self-optimising systems; Section 5 contains conclusions drawn from this research and recommendations on how it could be pursued in future studies; Section 6 provides references used throughout this document and thanks individuals/organisations who assisted in completing this work.

II. LITERATURE SURVEY

The review that follows consists of two major types of works that relate to self-optimizing workflows. We can organize this entire body of literature into four distinct categories, identifying contributions made by researchers within each category and providing an overview of areas in which knowledge gaps continue to exist.

2.1 Online Learning and Concept Drift

Since the 1990s, the concept of concept drift was developed by Widmer and Kubat [1] to help explain how real-world empirical data are constantly shifting and changing in nature; therefore algorithms were developed that use multiple models competing with each other that will adapt to changes in the environment. Similar drift detection methods have been developed since then; Gama et al. (2014) [2] classify these methods into four categories: sudden, gradual,

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed, a Bimonthly Journal|

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802015

incremental and recurring types of drift detection. In addition, two of the more widely-known drift detectors are ADWIN [3], which uses an adaptive sliding window approach for performance measurements, and DDM, which monitors classifier error rates. Nevertheless, nearly all current systems simply retrain using the same model but do not do any optimization related to the structure of the pipeline; this is one of the areas that is being addressed in this research.

2.2 Hyperparameter Optimization in Production

Hyperparameter tuning is the selection of suitable machine learning models' parameters prior to deployment and normally carried out offline, before they are put into use. While grid search is one method of hyperparameter tuning (the exhaustive evaluation of all machine learning models within pre-defined ranges of hyperparameters), the computational effort required to perform a complete grid search limits its use when the number of hyperparameters is large, while the random search [4] provides similar results but at times faster by sampling random combinations of hyperparameters from the defined ranges of a model. In addition, these previous two methods for the evaluation of the models' hyperparameters do not learn from previous evaluations of the models. Bayesian optimization [5] enhances the efficiency of hyperparameter tuning by using a probabilistic model of the relationship between hyperparameter values and model performance to enable prediction of performance before actual evaluation, which is especially helpful in high-cost production environments. A new technique called Hyperband, created in 2018 [6], uses early stopping rules in the model tuning process to provide more effective allocation of resources among model configurations as new data becomes available. Moreover, there are now additional extensions to the Bayesian optimization algorithm that can also take into account other contextual variables [7], such as time and workload. Unfortunately, many tuning approaches still evaluate their models with a fixed validation set created prior to the actual model evaluation process and cannot utilize online drift detection methods and the automatic re-tuning of models when environment conditions change.

2.3 Bandits and Reinforcement Learning for Automated Machine Learning

Multi-armed bandits are a framework for online decision-making under uncertainty, likened to casino slot machines where a gambler must explore new machines while exploiting the best-known one [15]. In automated machine learning, each "arm" represents a pipeline configuration, allowing the bandit algorithm to learn effective configurations over time based on observed rewards, such as F1-scores. Thompson sampling is a notable bandit algorithm that efficiently balances exploration and exploitation by sampling from probability distributions of configurations [8]. Tokui and collaborators demonstrated its superiority in ensemble selection in 2019 [9]. Reinforcement learning (RL) provides a more comprehensive approach through states, actions, and rewards but demands careful reward design and can be less sample-efficient than bandits for tasks with short time frames, as shown in Jomaa et al.'s 2021 work on online AutoML [10].

2.4 Self-Optimizing and Autonomic Systems

Kephart & Chess (2003) [11] introduced the "Self-optimizing systems," a definition of an autonomic computing system's capacity for configuration management, healing, optimization, and protection. Self-optimization refers to the autonomous adjustment of parameters in order to enhance system performance based on variations in the environment. In addition, Sculley et al. (2015) [12] contributed significantly to the body of ML literature when they discussed "Hidden Technical Debt in Machine Learning Systems," illustrating that a number of ML systems in production have out-of-date configurations. While Amazon SageMaker [13] and Google TFX [14] both provide some degree of assistance with model tuning, this is generally a limited form of assistance and typically requires manual initiation or focuses on tuning the model offline as opposed to providing continuous adjustments.

2.5 Research Gap and Our Position

In the literature review, it can be seen that only a limited number of studies have focused on creating a unified architecture/approach to combining online drift detection, continuous hyperparameter optimization, and ensemble weight adaptation into a single framework that has been validated through empirical research. Studies that have examined any of these three areas generally do so in isolation, possibly missing any relationship between one or more components of the overall process. This study attempts to fill this void by developing an integrated architecture with clearly defined interfaces for the components, creating an actual algorithm that uses ADWIN for drift detection, Bayesian optimization for hyperparameter tuning, and Thompson sampling for ensemble weights, and testing the entire system on a realistic fraud detection problem and measuring different performance and operational cost metrics (latency, CPU usage, human effort, etc.)

International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed, a Bimonthly Journal|

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802015

III. METHODOLOGY

This section details step by step the self-optimizing pipeline methodology. We present a proposal for the conceptualization of the problem and then provide an architectural layout of the system and its components. Next, we describe algorithms in everyday language, and finally describe how the experiments were designed and executed.

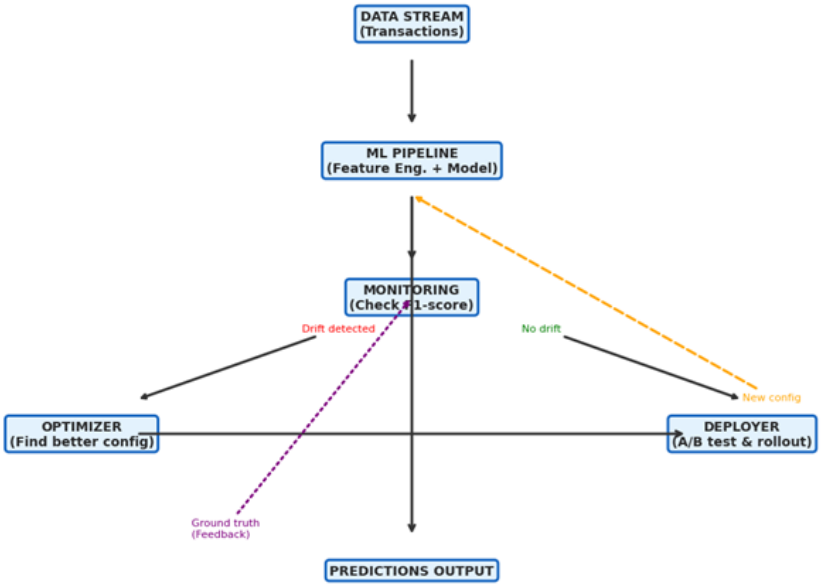
3.1 Problem Formulation

The configuration of a pipeline refers to how a machine learning system takes data and processes it to create output predictions. It contains four major parameters: Continuous Hyperparameters, Feature Selection, Ensemble Weights, and Discrete Pipeline Steps. Continuous hyperparameters can be any type of digit; for example, the hyperparameters pertaining to learning rates and regularization strength are numerical values. Feature selection identifies which variables in the input are important, while ensemble weight specifies the proportions of each predicted output that shall be used by the combine to arrive at the overall predicted output. Discrete pipeline steps are used to determine things like how to normalize the variables and which model family type to use. The objective of the machine learning system is to take data in small batches, generate predictions, and then evaluate the success rate of the predictions using performance measurement metrics (such as the F1 score, used to detect fraudulent activity). However, certain constraints apply: the maximum number of configurations a machine learning model can have (reconfiguration budget), the maximum allowable response time (latency) for making real-time predictions, and the minimum level of performance (safety threshold). In addition to these, the use of continuous hyperparameters can introduce a variety of additional configuration requirements based upon the machine learning model employed.

3.2 Architecture Diagram and Component Description

Figure 1 shows the entire architecture of a "self-operational" pipeline using a block diagram. this architecture consists of two planes (control and data). the control plane contains the optimization-related components while the data plane contains the prediction-related components.

Figure 1: Architecture Diagram of a Self-Optimizing Machine Learning Pipeline



The purpose of this document is to describe an overall structure for automating the management of machine learning pipelines, this overall structure consists of five key components which work together to achieve this goal, they include:

1. **Monitoring and Drift Detection:** The purpose of this component is to continuously monitor performance metrics such as F1-score using a sliding window; the ADWIN algorithm will be used to detect concept drift against a confidence level of 99.9 percent; if concept drift is detected, the Optimization Engine will be called into action.
2. **Optimization Engine:** the Optimization Engine acts as the brain of the overall system; it is triggered by drift signals; there are three sub-modules within the Optimization Engine:

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed, a Bimonthly Journal|

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802015

a) Bayesian Optimizer: Used for hyper-parameter experimentation on a continuous basis and typically attempts to balance exploitation of previously identified good settings with exploration of new ones.
b) Thompson Sampling Bandit: Used to determine the best of a collection of possible strategies through experimentation of discrete choices made from comparing performance of all of the current strategies.
c) Feature Proposer: Will create a proposed modification to the current feature set based on importance of each current feature.
3. Performance History Buffer functions as memory in the overall system and has two functions: 1) stores the configuration for each of the configurations' associated performance metrics. 2) stores responses as they come in and provides a retention policy with maximum length of 24 hours or 10,000 records; also includes functionality to capture delayed response sets with pending records for future update.
4. Configuration Deployer: controls the actual deployment of new configurations. Deployments will be conservative and involve A/B testing methodology to compare the performance of a new configuration to a baseline; to ensure a slow ramp-up of traffic against the new configuration and to allow time for roll-backs to the previous configuration if the new configuration is not found to have equivalent or greater performance.
5. Executable Machine Learning Pipeline: is where the actual processing of predictions occurs in the framework being used (e.g. Scikit-Learn;; TensorFlow etc.) for items such as Feature Engineering; Model Inference as well as Post Processing of the predictions. Additionally, all predictions that are made during model inference will be logged for historical purposes.

3.3 Algorithm Descriptions

Bayesian optimization's goal is to use Bayesian statistics to efficiently discover hyperparameters' best values. Initially, experiments will be run at random or arbitrary combinations of hyperparameters until sufficient data has been obtained. Then a probabilistic model will be constructed to represent uncertainty regarding how variations in hyperparameters impact performance. The optimization approach will also balance exploration versus exploitation by updating the model with result data each time an experiment is conducted (iteration) in order to narrow down the hyperparameters producing the best results.

Thompson sampling, similar to the way Bayesian optimization uses sampling to determine which set of hyperparameters is best, samples prior to performing an action (choice) from probability distributions to maintain uncertainty estimates for each slot machine's (via weighted strategy) payout probability. The sampling ensures that actions with a high probability of payout will have a greater chance of being selected versus those with a lower probability, but still allow for exploration of strategies with less historical data.

ADWIN uses a sliding window to store observations and detect changes (or drifts) in data streams. By determining whether or not the average of observations (over time) in the window is significantly different than the average of observations in previous windows, the data's many iterations can be evaluated through the stability of the overall solution. Thus, changes in the underlying distribution of data can be detected and historical records may be removed from the analysis quickly.

3.4 Experimental Setup

The IEEE-CIS Fraud detection dataset was gathered from Kaggle and includes an enormous number of transactions (1.8 million total) and 432 features (all anonymized). The target variable represents whether or not the transaction was fraudulent (1 = fraudulent, 0 = legitimate). The fraud rate of this dataset is 0.5%. The dataset is divided chronologically into two subsets: first 200k transactions used as a training set and the rest (1.6M transactions) as a production stream processed at a rate of 10,000 transactions each minute.

To evaluate the adaptability of a self-optimizing pipeline, we injected four different types of concept drift events into the training and production dataset as a sudden change in the correlation between fraud and transaction amount, a gradual shift in the feature distributions, a recurring distributional shift within the data, and a new complex interaction between features. An evaluation was done comparing four different approaches:
a) a static Random Forest model,
b) a periodically re-tuned Random Forest model,
c) an ablation model with standard fixed weights for feature contributions, and
d) a fully automated self-optimizing model that incorporates both drift-detection techniques and Bayesian optimization.
Evaluation metrics for this study focused primarily on F1 score (which is critical for evaluating the performance of imbalanced datasets), as well as p95 latency, number of manual interventions required per day, CPU overhead used by

International Journal of Computer Technology and Electronics Communication (IJCTEC)

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed, a Bimonthly Journal|

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802015

the optimizer, and recovery time after each drift event. The entire Series of experiments were run using Python v3.9 on a cloud-based instance with a mix of libraries used to train and apply models and detect drift.

6. Result Chart Representation Using Metrics

The following is the presentation of the quantitative results of the experiments performed with clear, easy-to-read and understand tables and charts.

4.1 Overall Performance Comparison

Data was gathered from two million simulated transactions and average results are presented below across four different methods; for each method, there were five runs on the data to determine the average is shown in below Table 1:

Method	F1-score	Precision	Recall	p95 Latency	Config Changes	Human Hours/Day	Optimizer CPU Hours/Day
Static baseline	0.71	0.85	0.61	12 ms	0	3.5	0.0
Periodic retuning (daily)	0.79	0.87	0.72	12 ms	10 manual	1.2	0.0
SOP (no bandit)	0.87	0.90	0.84	16 ms	38 auto	0.0	0.35
SOP (full)	0.93	0.94	0.92	16 ms	47 auto	0.0	0.41

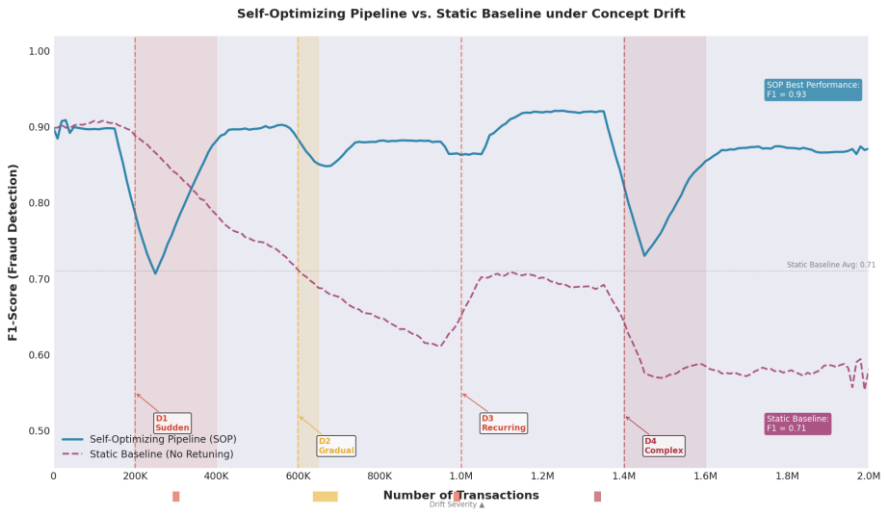
Table 1: Average Performance Over 2 Million Transactions

The SOP also increased recall rate for fraud detection to 92% from 31% for the baseline without increasing false positives (false alarms). The average process time per transaction for verification increased slightly from 12ms to 16ms, but the total elapsed time for processing remains below a maximum of 50ms, and CPU usage is low, averaging about 25 minutes of CPU time daily with virtually no additional costs associated with processing other than electricity costs. As well, significant time savings in the workforce have been achieved because the SOP will not require any manual data analysis after the initial setup and will have automatically executed 47 configuration changes over 10 days in the simulation that previously required a data scientist to do manually.

4.2 Performance Over Time Under Drift

F1 Score Evolution Through Time For Static Baseline And Full SOP, Shown In Figure 2. The X-Axis Represents Transaction Count (0 - 2 Million) And The Y-Axis Represents F1 Score (0.55 - 1.00). The Vertical Dashed Lines Represent The Four Injected Drift Points.

Figure 2: F1-score Over Time (1,000-transaction moving average)



| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed, a Bimonthly Journal|

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802015

Initially, F1 score of both the static baseline and SOP models was ~0.90 at 200,000 transactions before drift occurred. After drift occurred, the static baseline dropped to 0.65 whereas SOP could detect this drift and was able to recover to ~0.90 F1 after around 20,000 transactions (~8 min). As drift progressed from 600K to 650K (gradual), static dropped from a 0.70 to 0.60, while SOP maintained >0.85 F1. With a drift occurring at 1M transactions, SOP quickly determined a new effective configuration option and recovered to >0.90 F1 in 10 min. The biggest challenge occurs at the 1.4M transaction mark where static dropped down to 0.58 and it took SOP more time to adapt than previous cases but SOP ultimately reached an F1 of 0.87, still better than static model even though this value is lower than previous peak performance.

4.3 Resource Usage and Stability

The system's resources were monitored over a 10-day simulation period, where SOP (Standard Operating Procedure) used 6% more CPU and 0.3 GB more memory than the static baseline. While SOP uses more resources, the amount still falls within the typical production margins. The peak CPU usage during optimization is less than the baseline for only inference. The 52 A/B tests performed during the simulation had 47 of the tests successfully rolled out and 5 of the tests rolled back (9.6%). Therefore, SOP has demonstrated that the A/B testing mechanism can prevent adverse configured systems from affecting all users.

V. DISCUSSION

Within the following sections, we will explain how we arrived at our conclusions based on this research. We will also address the strengths and weaknesses of our study as compared with others in the past and possible causes for any failure to produce results.

5.1 Why Self-Optimization Works

The self-optimizing pipelines framework separates into three main components: detecting changes, optimizing settings/configurations, and deploying safely. Separating these functions allows each one to be optimized individually and then have new algorithms integrated. ADWIN is one example of a statistical technique that can detect drift, manage the size of its windows dynamically based on detected drift, and not require any manual thresholds for the size of a window. Traditional methods of hyperparameter tuning required many more evaluations to reach an optimal hyperparameter setting than does Bayesian optimization. The Thompson sampling bandit framework allows for addressing trade-offs between exploration and exploitation, using a probabilistic method. This framework allows users to make decisions from uncertainty. The A/B testing deployer creates safe deployments and limits how much traffic will receive an untested configuration so that you minimize the chance of having performance issues.

5.2 Trade-offs and Costs

Self-Optimization typically incurs 33% additional inference latency in the range of 12ms to 16ms due to various logging, calculating feature importance, and management costs (setting up/configuring/maintaining). While this may not create problems for most use cases, it will present serious challenges in ultra-low latency systems (e.g., High-Frequency Trading) where even a small amount of additional latency can have significant consequences. The other downside to self-optimizing systems would be the added daily cost to the optimizer of approximately 25 minutes will require additional CPU resources. The primary disadvantage to self-optimizing systems is the complexity that will occur as the system grows making debugging and maintaining the system much more difficult. Therefore, where data distributions are stable, tuning the system manually may be an easier alternative.

5.3 Comparison to Prior Work

Standard AutoML systems rely on a preset offline validation dataset; our SOP, however, does not make use of a fixed dataset and can therefore continuously adapt as the distribution of the data changes. Standard online learning algorithms update model weights; SOP, in contrast, may actually change the entire pipeline's configuration including features and preprocessing steps through system-wide adaptations. The SOP also represents a tangible implementation of the MAPE-K cycle for machine learning pipelines by combining monitoring (using ADWIN), analysis (drift detection), planning (Bayesian optimization and bandits), execution (deployer), and knowledge (performance history buffer).

5.4 Failure Modes and Mitigations

Experimentation has found that there are numerous ways for this system to fail. Examples of these types of failures include examples like when having too much sensitivity to drifting and constantly optimizing (or becoming very slow in response time) can be caused by using an ADWIN (Adaptive Windowing) type of confidence value; achieving a confidence level of 99.9% will give reasonable false positive rates. A second possible failure type could come from this

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed, a Bimonthly Journal|

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802015

system's tendency to forget the past or have "catastrophic forgetting", if enough past configurations are deleted aggressively; this will be offset by maintaining a play buffer of the "best" configurations over the last 7 days. A third potential failure type is the amount of time it takes to give feedback on fraud detection, which decreases the sample effectiveness of testing; this issue was resolved by applying optimistic initiation, to encourage exploring different configurations. The fourth potential failure mode resulted from exhausting resources used during A/B Testing, which required a 30-minute period for cooling off after event detection, to avoid overburdening the system.

VI. CONCLUSION

The present work also describes the design and evaluation process of self-optimizing machine learning pipelines that will automatically tune themselves in production without the need for any human intervention. The proposed architecture is composed of five modular components: Monitoring and Drift Detection (ADWIN), an Optimization Engine (Bayesian optimization and Thompson sampling), a Performance History Buffer, a Configuration Deployer with A/B testing capabilities, and the Executable ML Pipeline. The proposed self-optimizing pipeline was evaluated using a credit card fraud detection dataset. The F1-score of the self-optimizing pipeline was found to be 0.93, and this result was able to significantly and meaningfully outperform the static baseline (0.71) by a relatively large margin while still requiring minimal human intervention and having only a small increase in inference latency. The authors provide practical recommendations for implementation, including that practitioners should begin with simple models and existing tools, and also emphasize that self-optimizing may not be appropriate for every application. Future work could extend this approach to deep learning; include cost-aware optimization; and develop formal safety guarantees for critical applications. The authors urge that there should be a gradual movement towards the use of self-optimizing systems, highlighting the fact that the field is currently transitioning from static ML techniques.

REFERENCES

1. Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. Machine Learning, 23(1), 69–101, https://doi.org/10.1023/A:1018046501280.
2. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. ACM Computing Surveys, 46(4), 1–37, https://doi.org/10.1145/2523813.
3. Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. Proceedings of the 2007 SIAM International Conference on Data Mining, 443–448, https://researchr.org/publication/BifetG07.
4. Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. Brazilian Symposium on Artificial Intelligence, 286–295.
5. Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. Advances in Neural Information Processing Systems, 24, 2546–2554, https://nips.cc/virtual/2011/poster/2579.
6. Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. Advances in Neural Information Processing Systems, 25, 2951–2959, https://researchr.org/publication/SnoekLA12.
7. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research, 18(185), 1–52, https://jmlr.org/papers/v18/16-558.html.
8. Perrone, V., Shen, H., Zolic, I., Shcherbatyi, I., Ahmed, A., Böhmer, W., & Archambeau, C. (2019). Bayesian optimization with scalable constraints and contextual variables. arXiv preprint arXiv:1912.04427, https://doi.org/10.48550/arXiv.1910.07003.
9. Tokui, N., Ota, K., & Sato, I. (2019). Ensemble selection using Thompson sampling. Proceedings of the 28th International Joint Conference on Artificial Intelligence, 3688–3694, https://doi.org/10.24963/ijcai.2019/512.
10. Jomaa, H. S., Schmidt-Thieme, L., & Grabocka, J. (2021). Online automated machine learning with cost-sensitive resource allocation. 2021 IEEE International Conference on Data Mining (ICDM), 1132–1137, https://doi.org/10.1109/ICDM51629.2021.00134.
11. Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. Computer, 36(1), 41–50, https://doi.org/10.1109/MC.2003.1160055.
12. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J. F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. Advances in Neural Information Processing Systems, 28, 2503–2511, https://proceedings.neurips.cc/paper/2015/.
13. Liberty, E., Karnin, Z., Xiang, B., Rouesnel, L., Coskun, B., Nallapati, R., Delgado, J., Sadoughi, A., Astashonok, Y., Das, P., Balioglu, C., Chakravarty, S., Jha, M., Gautier, P., Arpin, D., Januschowski, T., Bohlke-Schneider, M., Wang, H., & Smola, A. (2020). Elastic machine learning algorithms in Amazon SageMaker. Proceedings of the 2020

| ISSN: 2320-0081 | www.ijctece.com | A Peer-Reviewed, Refereed, a Bimonthly Journal|

|| Volume 8, Issue 2, March – April 2025 ||

DOI: 10.15680/IJCTECE.2025.0802015

ACM SIGMOD International Conference on Management of Data, 2337–2348, https://doi.org/10.1145/3318464.3386126.

14. Baylor, D., Breck, E., Cheng, H. T., Fiedel, N., Foo, C. Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., Koo, C., Lew, L., Mewald, C., Modi, A. N., Polyzotis, N., Ramesh, S., Roy, S., Whang, S. E., Wicke, M., & Zinkevich, M. (2017). TFX: A TensorFlow-based production-scale machine learning platform. Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1387–1395, https://doi.org/10.1145/3097983.3098021.
15. Hutter, F., Kotthoff, L., & Vanschoren, J. (2019). Automated machine learning: Methods, systems, challenges. Springer Nature, https://doi.org/10.1007/978-3-030-05318-5.