Journal of Engineering Excellence
ISSN: **3108-2793** | Volume 2 Issue 4 October-December 2026
www.jeejournal.com **JEE-01**

IntelliLayer: A Deep Learning Architecture for Dynamic Decision Intelligence

[*1]**Gundeti Mounika**, *Guest lecturer,*
TGSWRDC(W), Adilabad, Telangana, India E-mail ID: gmounika111@gmail.com

[*2]**Vimula Shwetha,** *Assistant Professor, School of CSE, Department of CSE*
Malla Reddy (MR) Deemed To Be University, Medchal-Malkajgirii, Telangana, India
E-mail ID: shwethavimula9@gmail.com

[*3]**Dr K Sridhar,***Associate Professor, School of CSE, Department of CSE,*
MallaReddy(MR) Deemed to be University, Medchal,Telangana-500100, India
E-mail ID: sridhark529reddy@gmail.com

***Abstract*:** Organizations increasingly use data-driven systems to support operational decision-making in dynamic environments. However, many existing deep learning models focus primarily on predicting outcomes rather than recommending reliable actions. These models often provide limited information about prediction confidence, may experience performance degradation when data patterns change, and offer insufficient explanations for their decisions. Previous studies have addressed these challenges independently through improved prediction models, uncertainty estimation, calibration techniques, data drift detection, and explainable AI methods. To address these limitations, this paper introduces IntelliLayer, a five-layer deep learning architecture that combines these capabilities within a unified decision-making framework. The proposed architecture integrates static data and sequential events using Transformer encoders and a learnable gated fusion mechanism. It estimates calibrated outcome distributions through a deep ensemble and selects actions based on expected utility, while uncertain cases are referred to human reviewers. To handle changing data patterns, the framework employs adaptive windowing for drift detection and updates the model using replay techniques and elastic weight consolidation. Additionally, integrated gradients are used to explain individual predictions, and an audit record is maintained for every decision to support transparency and accountability. The paper describes the architecture of each layer, outlines the online decision-making algorithm, and presents a reference implementation stack. It also defines a reproducible evaluation framework covering predictive performance, calibration, decision-making costs, selective prediction coverage, recovery from data drift, and processing latency. Since experimental results have not yet been obtained, the proposed evaluation protocol and results template provide a structured basis for future validation of the IntelliLayer architecture.

Keywords—decision intelligence; deep learning; uncertainty quantification; concept drift; continual learning; explainable AI.

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

1. INTRODUCTION

Decision intelligence focuses on developing systems that convert data into meaningful actions rather than simply predicting future outcomes. Applications such as credit approval, inventory management, fraud detection, maintenance planning, and clinical prioritization all involve a similar process. Each case must be evaluated, an appropriate action must be selected based on its possible consequences, and the decision must be explained to the person responsible for it. Duan et al. [1] highlight the integration of artificial intelligence with human decision-making and the circumstances in which AI should assist or replace human judgment as important challenges in information systems research.

Deep learning has significantly improved the ability of machines to learn useful representations from data, particularly in areas such as image recognition and natural language processing [2]. Its use has also expanded to structured business data and operational applications. Despite these advances, many practical systems remain focused on generating predictions rather than supporting complete decision-making processes. Four major challenges distinguish a predictive model from a reliable decision-support system.

1. **Prediction does not automatically lead to a decision.** A classification model typically produces a category or probability score. However, selecting the best action requires considering the potential costs, benefits, and risks associated with each available option. Many existing systems rely on fixed decision thresholds that are established during development and may not remain suitable under changing conditions.
2. **Prediction confidence may be unreliable.** Modern neural networks can produce highly confident predictions even when their outputs are incorrect. In particular, softmax probabilities may be poorly calibrated, making it difficult to determine whether a prediction is sufficiently reliable for practical use.
3. **Real-world data patterns change over time.** Customer preferences, market conditions, product prices, and fraudulent activities continually evolve. Consequently, models trained on historical data may lose accuracy when the relationship between input features and outcomes changes. Such changes, commonly referred to as data or concept drift, present significant challenges for maintaining model performance [3], [4].
4. **Decisions require clear explanations.** Regulatory requirements and operational practices often demand understandable reasons for individual decisions. When a model operates as a black box, domain experts may find it difficult to verify its outputs, identify potential errors, or establish confidence in its recommendations [5].

Although these challenges have been widely investigated, existing approaches often address them separately. Advanced models for tabular and sequential data focus mainly on predictive performance [6], [7], while uncertainty estimation and calibration methods concentrate on measuring the reliability of predictions. Similarly, drift detection and adaptation techniques [3], [4] are often evaluated independently of complex deep learning architectures. Explainability methods are also frequently applied after predictions have been generated, rather than being integrated into the decision-making process and its associated records. As a result, practitioners seeking to combine these capabilities must often develop separate components without a unified framework that explains how they work together.

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

To address this limitation, this paper introduces **IntelliLayer**, a layered deep learning architecture designed to integrate predictive performance, uncertainty estimation, adaptive learning, and explainability into a unified decision pipeline. The proposed framework aims to make each capability a distinct and replaceable component with a clearly defined interface. The research objectives are to (i) develop a modular architecture that combines these capabilities, (ii) establish a formal description of action selection, human deferral, drift response, and decision explanations, and (iii) define an evaluation methodology that assesses practical decision quality in addition to conventional predictive accuracy.

The main contributions of this work are as follows:

- **Unified architecture:** A five-layer deep learning framework that combines static and temporal data representations through gated fusion, estimates uncertainty using an ensemble of models, selects actions according to expected utility, refers uncertain cases to human reviewers, and supports continual learning when data drift is detected.
- **Formal specification:** A mathematical description of the individual layers, including the equations governing feature fusion, action selection, uncertainty-based deferral, and model adaptation, together with an algorithm for online decision-making.
- **Reference implementation:** A proposed implementation stack based on established open-source libraries to support the development and practical evaluation of the architecture.
- **Reproducible evaluation framework:** A structured evaluation protocol that includes baseline comparisons, ablation studies, decision-oriented performance metrics, and a standardized template for reporting experimental results.

The remainder of this paper is organized as follows. Section 2 reviews the relevant literature and existing approaches. Section 3 introduces the IntelliLayer architecture and its mathematical formulation. Section 4 describes the implementation strategy and operational workflow. Section 5 presents the proposed evaluation protocol and discusses anticipated trade-offs and limitations. Finally, Section 6 summarizes the work and outlines possible directions for future research.

2. Literature Review

Previous research related to IntelliLayer can be grouped into five major areas: deep learning for structured data, decision-making methods, uncertainty estimation, adaptation to changing data distributions, and explainable AI. Each area contributes useful techniques, but no single approach fully addresses all the requirements of a reliable decision-support system. Table 1 provides a comparison of these research areas based on the capabilities they offer.

2.1 Deep Learning for Tabular and Temporal Data

Arık and Pfister (2021) introduced TabNet [6], a deep learning model that applies sequential attention to select relevant features at different decision steps. Its feature-selection masks help identify which input variables influence individual predictions, while masked self-supervised pretraining can improve performance when sufficient unlabeled data are available. However, TabNet processes records without explicitly modeling temporal relationships and does not inherently provide calibrated uncertainty estimates or mechanisms for handling concept drift.

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

Grinsztajn et al. (2022) [8] compared neural networks with tree-based methods across medium-sized tabular datasets. Their findings showed that tree-based ensembles continued to outperform most neural approaches on many of the evaluated datasets. Therefore, gradient-boosted tree models remain important baselines when assessing the performance of deep learning architectures for structured data.

Lim et al. (2021) developed the Temporal Fusion Transformer (TFT) [7], a model designed for forecasting across multiple future time steps. It combines variable-selection networks, recurrent layers for local temporal patterns, interpretable multi-head attention for capturing long-range dependencies [9], and gating mechanisms that control the contribution of different components. The architecture can process static features together with time-varying inputs and generate quantile-based forecasts. These characteristics make TFT a relevant foundation for the representation layer of IntelliLayer. However, TFT primarily focuses on forecasting future values rather than selecting actions, and it does not inherently provide a complete mechanism for responding to changes in the underlying data distribution.

2.2 Learning Decisions

Mnih et al. (2015) [10] demonstrated that deep reinforcement learning can learn control policies directly from reward signals. Although this approach is powerful for sequential decision-making, it often requires substantial interaction with an environment or a reliable simulator. In practical applications such as lending and healthcare, exploratory actions may introduce unacceptable financial, operational, or safety risks.

In contrast, many organizational decisions can be modeled as individual action-selection problems in which the possible outcomes and their associated costs or benefits are known or can be estimated. In these situations, a predictive model combined with an explicit utility function offers a practical alternative. Such a framework makes it easier to examine how decisions are made and to update action-selection rules when business priorities or operational costs change. IntelliLayer follows this approach by combining outcome predictions with expected utility to determine suitable actions.

2.3 Uncertainty Estimation and Calibration

Gal and Ghahramani (2016) [11] demonstrated that dropout applied during inference can approximate Bayesian inference and provide uncertainty estimates without requiring a separate probabilistic model. Lakshminarayanan et al. (2017) [12] introduced deep ensembles, in which multiple neural networks are trained with different initializations and their predictions are combined. Their experiments demonstrated that this approach can provide useful uncertainty estimates and improve robustness under certain forms of dataset shift.

Guo et al. (2017) [13] investigated the calibration of modern neural networks and found that many models tend to produce overconfident predictions. They demonstrated that temperature scaling, a post-processing technique involving a single adjustable parameter, can substantially improve calibration. These methods provide important tools for estimating prediction reliability. However, uncertainty estimation and calibration alone do not determine how a system should respond to uncertain predictions. IntelliLayer addresses this limitation by using

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

calibrated outcome distributions to support action selection and by allowing uncertain cases to be referred to human reviewers.

2.4 Learning Under Concept Drift

Gama et al. (2014) [3] reviewed methods for adapting to concept drift and classified them according to their approaches to memory management, change detection, learning, and error estimation. Lu et al. (2019) [4] further organized the field around three main activities: detecting changes, understanding their causes or characteristics, and adapting models to the new conditions.

Bifet and Gavaldà (2007) [14] proposed ADWIN, an adaptive sliding-window algorithm that adjusts the amount of retained data according to observed changes in the data stream. The window can expand when the underlying process remains stable and contract when statistical changes are detected. This approach provides a foundation for monitoring evolving data distributions.

However, updating a deep learning model using only recent observations can cause catastrophic forgetting, in which previously learned knowledge is lost. Kirkpatrick et al. (2017) [15] introduced elastic weight consolidation (EWC), a regularization method that reduces changes to parameters considered important for previously learned tasks. Parisi et al. (2019) [16] reviewed several continual-learning strategies, including regularization, replay-based learning, and architectural approaches. Building on these ideas, IntelliLayer combines drift detection with replay and EWC to support model adaptation while attempting to preserve useful knowledge from earlier data.

2.5 Explainability

Lundberg and Lee (2017) [17] introduced SHAP, a unified framework for feature attribution based on Shapley values from cooperative game theory. The method provides a consistent way to estimate the contribution of individual features to a model's prediction. Sundararajan et al. (2017) [18] proposed integrated gradients, which attributes predictions to input features by integrating gradients along a path from a baseline input to the actual input. The method satisfies the sensitivity and implementation-invariance axioms under its stated formulation and requires access to the model's gradients.

Barredo Arrieta et al. (2020) [5] reviewed explainable artificial intelligence and emphasized its importance in developing transparent and responsible AI systems. These methods help users understand the factors influencing model outputs, but explanations are often generated separately from the operational decision process. IntelliLayer incorporates integrated-gradient explanations into its decision pipeline and associates them with individual decisions and audit records to improve traceability and support subsequent review.

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

2.6 Research Gap

Existing approaches address different aspects of intelligent decision-making, but none provides a complete solution that integrates all five required capabilities into a single framework. Combining these methods also involves more than simply connecting individual components, as their outputs and operations must work together effectively. For example, uncertainty estimates must directly influence action selection, drift detection must initiate model updates without losing previously acquired knowledge, and explanations must reflect the actual decisions made by the system. To address these challenges, IntelliLayer introduces a unified architecture with clearly defined interfaces between its components. Unlike approaches that primarily focus on predictive accuracy, the proposed framework also emphasizes practical decision quality by evaluating factors such as decision-making cost, prediction coverage, and the system's ability to handle changing data patterns.

Table 1. Comparison of representative approaches according to the capabilities they provide. The symbols ✓, Partial, and — indicate that a capability is provided, partially supported, or not directly addressed, respectively.

Approach	Static + temporal inputs	Uncertainty	Action selection	Drift adaptation	Explanation
TabNet [6]	— (static only)	—	—	—	✓
TFT [7]	✓	Partial (quantiles)	—	—	✓
Deep RL (DQN) [10]	Partial	—	✓	—	—
Deep ensembles [12] + temperature scaling [13]	—	✓	—	—	—
ADWIN [14]	—	—	—	Partial (detection)	—
EWC [15]	—	—	—	Partial (retention)	—
SHAP [17], integrated gradients [18]	—	—	—	—	✓
IntelliLayer (proposed)	✓	✓	✓	✓	✓

3. Architecture / System Design

IntelliLayer consists of five interconnected layers, as illustrated in Fig. 1. Each layer is designed with clearly defined inputs and outputs, allowing individual components to be modified, replaced, or evaluated independently through ablation studies. Incoming cases pass sequentially through the layers, beginning with data ingestion and ending with action

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

selection. Meanwhile, feedback from observed outcomes is directed back to the adaptation layer, enabling the system to update its model and respond to changes in data patterns over time.

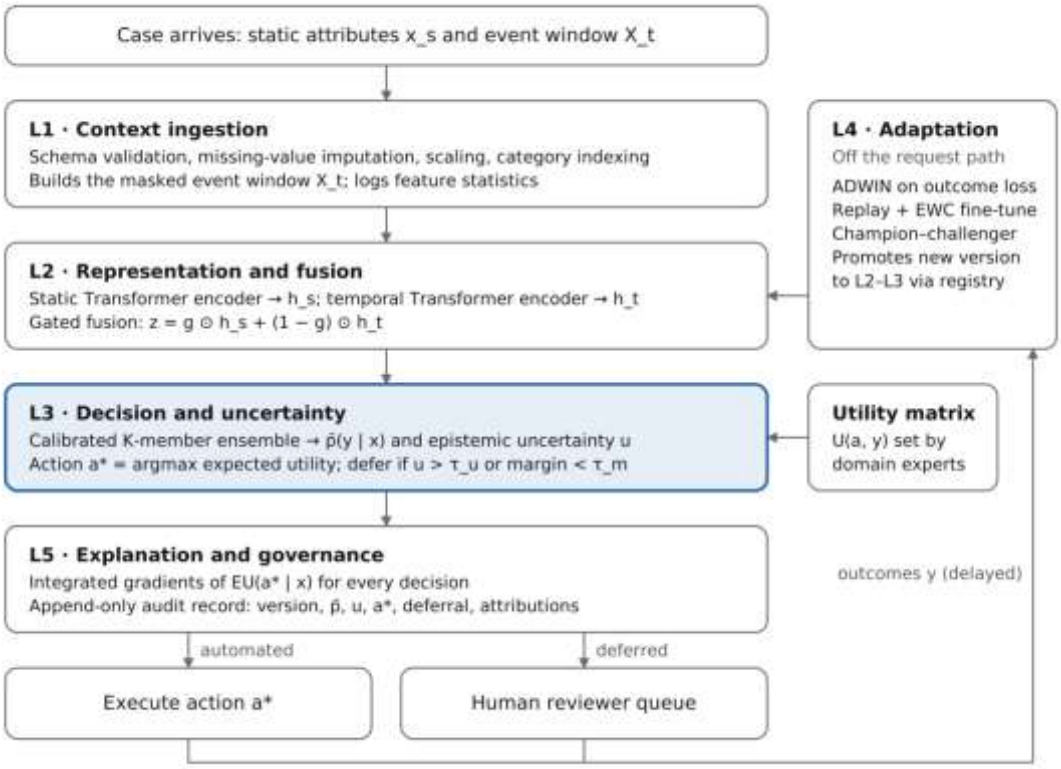


Fig. 1. IntelliLayer architecture. Layers L1, L2, L3, and L5 operate along the main request-processing path, while L4 functions alongside this path by processing delayed outcome feedback and deploying updated model versions to L2 and L3.

3.1 Problem Formulation

At time (t), each incoming case consists of static attributes (x_t^{(s)}), such as customer profile information, and a sequence of recent events (X_t^{(e)}), where each event represents an observation from the past. The system must select an action (a_t) from a finite set of available actions (\mathcal{A}). The actual outcome (y_t) belongs to a finite set of possible outcomes (\mathcal{Y}) and may only become available after a certain delay. A utility matrix (U), defined by domain experts, specifies the value associated with taking action (a) when the outcome is (y). This matrix can represent unequal costs, such as the consequences of missing a fraudulent transaction compared with incorrectly flagging a legitimate one. When the system cannot make a sufficiently reliable decision, it can defer the case to a human reviewer.

3.2 Layer 1: Context Ingestion

The context ingestion layer validates incoming records against a predefined schema and handles missing values using statistics calculated from the training dataset. Numerical features are standardized, while categorical variables are converted into integer indices for

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

subsequent processing. The layer also constructs a window of recent events, padding shorter sequences when necessary and maintaining a mask to distinguish valid events from padding. In addition, relevant feature statistics are recorded to support the monitoring of input-data drift during system operation.

3.3 Layer 2: Representation and Fusion

The representation layer processes static attributes and temporal event sequences through separate encoders before combining their outputs. In the static encoder, each numerical attribute (x_i) is transformed into a learned vector representation, while each categorical attribute is mapped to a trainable embedding. A Transformer encoder [9] processes these attribute representations together with a learnable summary token to produce the static representation (h_s).

For temporal data, the event features are projected into a common representation space and combined with positional encodings to preserve the order of events. The resulting sequence is processed by a masked Transformer encoder, which produces a summary representation (h_t). The masking mechanism prevents padded positions from contributing to the attention calculations.

Both encoders use scaled dot-product attention, as defined in Equation (1). In this formulation, (Q), (K), and (V) represent the query, key, and value matrices, respectively, while (d_k) denotes the dimensionality of the key vectors.

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)V \tag{1}$$

The static and temporal representations are combined using a learnable gating mechanism inspired by the gating layers introduced in the Temporal Fusion Transformer (TFT) [7]. As defined in Equation (2), the gate dynamically adjusts the contribution of each representation to the final fused feature vector, allowing the model to emphasize the most relevant information from both data sources.

$$\mathbf{g} = \sigma\big(W_g[\mathbf{h}_s;\mathbf{h}_t] + \mathbf{b}_g\big), \qquad \mathbf{z} = \mathbf{g} \odot \mathbf{h}_s + (\mathbf{1}-\mathbf{g}) \odot \mathbf{h}_t \tag{2}$$

Here, (\sigma) represents the logistic sigmoid function, (\oplus) denotes concatenation, and (\odot) indicates element-wise multiplication. The learned gate (g) dynamically controls the contribution of the static and temporal representations to the fused output. For cases with limited event histories, the model can place greater emphasis on static attributes, whereas cases with richer histories can benefit more from behavioral information. The average gate value for each case is also recorded as a coarse indicator of the relative contribution of the two representations to the final decision.

3.4 Layer 3: Decision and Uncertainty

The decision layer employs a deep ensemble [12] consisting of (M) independently initialized encoder–prediction-head networks. Each ensemble member (m) produces a vector of logits (z_m), which is converted into a temperature-scaled probability distribution (p_m(y\mid x)). The temperature parameter (T) is estimated using validation data to improve probability calibration [13]. The ensemble prediction and its associated epistemic uncertainty are defined in Equation (3).

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

Journal of Engineering Excellence
ISSN: **3108-2793** | Volume 2 Issue 4 October-December 2026
www.jeejournal.com **JEE-01**

$$\bar{p}(y \mid x) = \frac{1}{K}\sum_{k=1}^{K} p_k\,(y \mid x), \qquad u_{\text{epi}}\,(x) = \mathscr{H}[\bar{p}] - \frac{1}{K}\sum_{k=1}^{K} \mathscr{H}[p_k] \tag{3}$$

Here, (H(\cdot)) denotes Shannon entropy. The total predictive entropy (H(\bar{p})) represents the overall uncertainty associated with the ensemble prediction, while the average entropy of individual ensemble members reflects the uncertainty inherent in the data. The difference between these quantities, expressed as (I(y; m \mid x)), represents the mutual information between the predicted outcome and the ensemble member, providing a measure of disagreement among the models. This disagreement can increase when the system encounters unfamiliar inputs or patterns that differ from its training data.

The system selects an action by calculating the expected utility of each available option using the predicted outcome probabilities and the predefined utility matrix. The action with the highest expected utility is then selected, as specified in Equation (4).

$$a^* = \underset{a\in\mathcal{A}}{\text{argmax}}\ \text{EU}(a \mid x), \qquad \text{EU}(a \mid x) = \sum_{y\in\mathcal{Y}} \bar{p}\,(y \mid x)\,U(a,y) \tag{4}$$

The system refers a case to a human reviewer when the model's uncertainty exceeds a predefined threshold or when the expected utility difference between the best and second-best actions is too small. This approach helps prevent potentially unreliable decisions from being made automatically, particularly when the model lacks sufficient confidence or when multiple actions have similar expected benefits. The deferral condition is formally defined in Equation (5).

$$\text{defer}(x) = \left[u_{\text{epi}}\,(x) > \tau_u\right] \vee \left[\text{EU}(a^* \mid x) - \text{EU}\big(a_{(2)} \mid x\big) < \tau_m\right] \tag{5}$$

The thresholds (\tau_H) and (\tau_\Delta) are determined using validation data to achieve a desired level of coverage, defined as the proportion of cases handled automatically by the system. Keeping the utility matrix separate from the learned model also allows business costs and priorities to be updated without retraining the model.

3.5 Layer 4: Adaptation

When the actual outcomes become available, the adaptation layer calculates the loss associated with each case and passes the resulting values to an ADWIN drift detector [14]. ADWIN maintains a window (W) of recent loss observations and monitors changes in their statistical behavior. A drift signal is generated when the window can be divided into an older segment (W_0) and a newer segment (W_1), containing (n_0) and (n_1) observations, respectively, and the difference between their estimated means exceeds the statistical bound defined by the algorithm. The bound is calculated using Equation (6), where (\delta) represents the confidence parameter that controls the detector's sensitivity to changes.

$$\varepsilon_{\text{cut}} = \sqrt{\frac{1}{2m}\ln\frac{4}{\delta'}}, \qquad m = \frac{1}{1/n_0 + 1/n_1}, \qquad \delta' = \frac{\delta}{|W|} \tag{6}$$

When a drift signal is detected, the adaptation layer fine-tunes a challenger model using a combination of recent observations and samples drawn from a replay buffer maintained through reservoir sampling [16]. To reduce the risk of catastrophic forgetting, the training process incorporates an elastic weight consolidation (EWC) penalty [15], which discourages

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

substantial changes to parameters that are important for previously learned patterns. The resulting objective function is defined in Equation (7), balancing adaptation to recent data with the retention of useful knowledge acquired from earlier observations.

$$\mathscr{L}(\theta) = \mathscr{L}_{\text{recent Ureplay}}(\theta) + \frac{\lambda}{2}\sum_{i} F_i(\theta_i - \theta_i^*)^2 \quad (7)$$

Here, (F_i) represents the diagonal Fisher information associated with parameter (\theta_i) at the current model parameters (\theta^*), while (\lambda) determines the strength of the EWC regularization penalty. The challenger model replaces the existing model only when it achieves a lower expected decision cost on a held-out set of recently labeled cases. After the update is accepted, the temperature parameter (T) and the deferral thresholds are recalibrated to reflect the updated model's behavior. This champion–challenger evaluation process helps prevent unreliable drift signals or unsuccessful updates from reducing the performance of the deployed system.

3.6 Layer 5: Explanation and Governance

For each decision, the explanation layer applies integrated gradients [18] to identify how individual input features contribute to the expected utility of the selected action. The method evaluates the gradients along a path from a baseline input to the actual input, allowing the system to estimate the contribution of each feature to the final decision. This process is formally defined in Equation (8), where the feature attributions help provide a clearer understanding of the factors influencing the selected action.

$$\mathrm{IG}_j(x) = \left(x_j - x'_j\right)\int_0^1 \frac{\partial\,\mathrm{EU}\left(a^* \mid x' + \alpha(x - x')\right)}{\partial x_j}\,d\alpha \quad (8)$$

The baseline input (x') is constructed using the training-set mean for numerical features and a learned neutral embedding for categorical features. When categorical variables are represented through embeddings, the attribution values across their embedding dimensions are aggregated to obtain a single contribution score for each original feature. By explaining the expected utility of the selected action rather than an individual class probability, the framework ensures that the explanation is directly connected to the decision made by the system. In addition, SHAP [17] summaries are generated offline to examine the model's overall behavior and identify influential features across multiple cases.

For traceability, every decision generates an append-only audit record containing the input hash, model version, predicted outcome distribution, uncertainty estimates, selected action, deferral status, and the most influential feature attributions. When available, the record also includes the human reviewer's decision and the actual outcome. This information supports subsequent auditing, performance analysis, and investigation of individual decisions.

3.7 Reliability, Scalability, and Security

The inference process is stateless when the event window is provided, allowing the serving infrastructure to scale horizontally according to demand. The ensemble members can also be evaluated in parallel to reduce processing time. Each model version is stored as an immutable artifact, making it possible to restore the previously deployed model if an update causes performance degradation.

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

To support security and accountability, the audit log is maintained in append-only form. Personally identifiable information is either excluded from storage or protected through hashing before records are retained, and access to reviewer actions is restricted through appropriate access controls. Although ensemble inference increases computational requirements approximately in proportion to the number of independently evaluated models, a shared-encoder architecture with multiple prediction heads offers a potentially more efficient alternative. This alternative is included in the proposed ablation study to assess the trade-off between computational cost and predictive reliability.

4. Implementation Techniques and Workflow

This section outlines a reference implementation of IntelliLayer using established open-source libraries. At the current stage, IntelliLayer is a proposed architecture rather than a deployed system. The components and workflow described below establish a practical foundation for implementation and future evaluation. No deployment results or measured performance improvements are claimed at this stage.

4.1 Software Stack

Table 2. Reference implementation stack

Layer	Function	Library or tool
L1 Ingestion	Schema validation, imputation, scaling, windowing	Python 3.10+, pandas, scikit-learn
L2 Representation	Static and temporal Transformer encoders, gated fusion	PyTorch 2.x
L3 Decision	Ensemble training, temperature scaling, utility and deferral	PyTorch, NumPy
L4 Adaptation	ADWIN drift detection; replay and EWC fine-tuning	river (ADWIN), PyTorch
L5 Explanation	Integrated gradients per decision; offline SHAP	Captum, SHAP
Serving and governance	REST inference API, model registry, audit store	FastAPI, MLflow, PostgreSQL

Training the ensemble members is expected to require GPU resources for practical execution times, whereas inference can be performed on either a CPU or a GPU. The exact hardware specifications and processing times will be documented during experimentation because they depend on factors such as the ensemble size (M), event-window length (L), and hidden-layer dimension (d).

4.2 Initial Configuration

The initial configuration provides a starting point for implementation, with the final hyperparameters to be selected using validation data. The proposed settings include a hidden dimension (d), two Transformer layers with four attention heads in each encoder, an event window of length (L), and an ensemble containing (M) independently initialized models, following the small-ensemble approach described in [12]. Model optimization will use

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

AdamW, with early stopping based on validation decision cost. The ADWIN detector will use a confidence parameter (\delta) set to the default value provided by the selected implementation. A replay buffer with a fixed capacity will be maintained and updated using reservoir sampling.

These settings represent initial design choices rather than experimentally established optimal values. Their suitability will be assessed during validation.

4.3 Offline Training Workflow

1. **Data preparation:** Historical cases are divided chronologically into training, validation, and test periods to prevent information from the future from influencing model development. Missing-value imputation and numerical scaling statistics are calculated exclusively from the training period.
2. **Ensemble training:** A collection of (M) encoder–prediction-head networks is trained using different random seeds. Each network minimizes the cross-entropy loss between its predicted outcome distribution and the observed target labels.
3. **Probability calibration:** The temperature parameter (T) is optimized on the validation period by minimizing the negative log-likelihood, following the temperature-scaling approach described in [13].
4. **Deferral threshold selection:** Candidate values for the uncertainty threshold (\tau_H) and the expected-utility margin threshold (\tau_\Delta) are evaluated on the validation dataset. The selected thresholds aim to minimize expected decision cost while satisfying a predefined automatic-decision coverage target, such as 90%.
5. **Adaptation initialization:** The diagonal Fisher information matrix (F) is estimated using the trained model parameters. The replay buffer is initialized with observations from the training period, and the selected model is registered as the current champion.
6. **Deployment:** The versioned model, utility matrix, calibration settings, and deferral thresholds are published to the serving infrastructure as a consistent deployment configuration.

4.4 Online Decision Algorithm

Algorithm 1 describes the online decision-making and adaptation workflow of IntelliLayer. Incoming cases are processed immediately through the inference pipeline, while model adaptation operates asynchronously as delayed outcome labels become available. This separation allows the system to continue serving requests while monitoring performance and preparing potential model updates.

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

Algorithm 1. IntelliLayer online decision-making and adaptation procedure.

```
Input: ensemble {f_1..f_K}, temperature T, utility U, thresholds tau_u, tau_m,
      ADWIN detector D, replay buffer B, Fisher F, champion theta*

on new case (x_s, X_t):
   x <- Ingest(x_s, X_t)                    # L1
   for k = 1..K in parallel:                # L2 + L3
p_k<- softmax( f_k( Fuse(Enc_s(x), Enc_t(x)) ) / T )
p_bar<- mean_k(p_k);   u <- H(p_bar) - mean_k(H(p_k))     # Eq. (3)
   for each action a in A:  EU[a] <- sum_yp_bar[y] * U[a, y] # Eq. (4)
   a* <- argmax_a EU[a];   margin <- EU[a*] - second_max(EU)
   if u >tau_u or margin <tau_m:  send case to reviewer queue   # Eq. (5)
   else:                    execute a*
attr<- IntegratedGradients(EU[a*], x)          # L5, Eq. (8)
AuditLog.append(hash(x), version, p_bar, u, a*, deferred, topk(attr))

on outcome y for logged case x:
   loss <- -log p_bar[y];   D.update(loss);   B.reservoir_add(x, y)
   if D.drift_detected():                   # L4, Eq. (6)
theta_c<- FineTune(theta*, recent + B, Eq. (7))
      if Cost(theta_c, holdout) < Cost(theta*, holdout):
         theta* <- theta_c;  refit T, tau_u, tau_m;  recompute F
```

4.5 Module Integration

The five layers communicate through structured records with clearly defined data types and interfaces. Layer L1 produces a validated tensor bundle containing the processed input features and an associated mask. Layer L2 generates the fused representation (h), which is passed to Layer L3. The decision layer then produces the predicted outcome distribution, uncertainty estimates, expected utilities, and either the selected action or a deferral decision. Layer L5 uses the case information and the resulting decision to generate the corresponding audit record.

Layer L4 operates independently of the request-processing path and interacts with the serving infrastructure through the model registry. This separation prevents retraining activities from directly affecting decision latency and allows individual components to be replaced without requiring changes to neighboring layers. For example, a TabNet-based static encoder [6] could replace the Transformer-based static encoder while preserving the defined interfaces between the components.

5. Results and Discussion

This version of the paper does not include experimental results. Instead, this section defines the proposed evaluation methodology, introduces a results template in Table 4, and outlines the hypotheses that future experiments should investigate. All entries marked with "—"

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

represent values that must be obtained through experimentation. No numerical values in this section should be interpreted as measured findings.

5.1 Experimental Setup

Datasets. The evaluation is planned around three types of data sources.

(a) Public concept-drift benchmarks, including the Electricity and Forest Covertype data streams discussed in [4], will be used to examine model adaptation under changing data distributions.

(b) A domain-specific dataset containing static attributes, event histories, delayed outcome labels, and a utility matrix defined with input from domain experts will be used to evaluate the complete decision pipeline. The dataset source, number of observations, class distribution, and utility values must be documented to ensure reproducibility.

(c) Controlled drift scenarios will be introduced into the test period of the domain-specific dataset. Both abrupt and gradual changes will be simulated at predetermined time points, allowing drift-detection delay and model recovery to be measured against known change points.

Evaluation protocol. The data will be divided chronologically into training, validation, and test periods. Conventional offline metrics will be calculated on the held-out test period. For streaming evaluation, prequential test-then-train evaluation [3] will be employed. Under this procedure, each incoming case is evaluated before it contributes to model updates, which occur only after the corresponding outcome becomes available.

Each experimental configuration will be evaluated using five random seeds. Results will be reported as mean values with standard deviations, and differences between IntelliLayer and the strongest baseline will be assessed using paired bootstrap confidence intervals.

Baselines. Five baseline configurations are proposed for comparison:

- **B1:** Gradient-boosted trees using a fixed decision threshold, representing a strong baseline for structured data based on the findings in [8].
- **B2:** A multilayer perceptron used as a conventional neural network baseline.
- **B3:**TabNet [6], which applies attention-based feature selection to tabular data.
- **B4:** A Temporal Fusion Transformer (TFT)-style forecasting model [7] combined with a threshold-based decision rule.
- **B5:** Gradient-boosted trees retrained from scratch whenever the ADWIN detector [14] identifies a potential drift event.

Ablation studies. The following configurations will be evaluated to determine the contribution of individual architectural components:

- **A1:** Replace gated fusion with simple concatenation of static and temporal representations.
- **A2:** Use a single network instead of a deep ensemble, thereby removing ensemble-based epistemic uncertainty estimation.
- **A3:** Replace the deep ensemble with Monte Carlo dropout [11] for uncertainty estimation.
- **A4:** Disable human deferral and require the system to make an automatic decision for every case.

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

- **A5:** Remove the adaptation layer to evaluate performance without drift-triggered updates.
- **A6:** Use ordinary fine-tuning without replay or elastic weight consolidation.
- **A7:** Replace independently trained ensemble encoders with a shared encoder and multiple prediction heads to examine computational efficiency.

5.2 Evaluation Metrics

Table 3 presents the metrics proposed for evaluating predictive quality, probability calibration, decision performance, selective prediction, adaptation, and computational efficiency. Calibration will be assessed using expected calibration error (ECE), calculated over (B) confidence bins containing the model's predictions for (N) cases, as defined in Equation (9).

$$\mathrm{ECE} = \sum_{b=1}^{M} \frac{|B_b|}{n} |\mathrm{acc}(B_b) - \mathrm{conf}(B_b)| \tag{9}$$

Table 3. Evaluation metrics

Property	Metric	Definition
Predictive quality	Macro-F1; negative log-likelihood	Class-balanced accuracy; quality of probabilities
Calibration	Expected calibration error (ECE)	Weighted gap between confidence and accuracy, Eq. (9)
Decision quality	Mean realized utility per case	Average $U(a, y)$ over automated decisions, with deferred cases charged a review cost
Selective prediction	Coverage; selective risk; area under the risk–coverage curve	Share decided automatically; cost on those cases; trade-off across thresholds
Adaptation	Detection delay; recovery time; forgetting	Cases between true and detected drift; cases until rolling cost returns within ϵ of its pre-drift level; cost increase on the earlier period after an update
Explanation	Deletion faithfulness	Drop in $\mathrm{EU}(a^* \mid x)$ when the top-k attributed fields are replaced by baseline values, compared with k random fields
Efficiency	Latency (p50, p95); throughput	Time per decision including attribution; decisions per second

Realized utility is treated as the primary evaluation metric because improvements in predictive accuracy do not necessarily translate into better operational decisions. A model may classify more cases correctly while still making costly mistakes, such as failing to identify fraudulent activity or incorrectly rejecting legitimate applications. Therefore, evaluating the actual utility of the actions taken provides a more meaningful assessment of the system's practical value.

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

5.3 Results Template

Table 4. Proposed results template for dataset (b). All entries will be completed using experimental measurements, with results reported as the mean and standard deviation across five independent random seeds.

Model	Macro-F1	ECE	Mean utility	Coverage (%)	Recovery time (cases)	p95 latency (ms)
B1 Gradient-boosted trees	—	—	—	100	—	—
B2 MLP	—	—	—	100	—	—
B3 TabNet	—	—	—	100	—	—
B4 TFT-style + threshold	—	—	—	100	—	—
B5 Trees + ADWIN retraining	—	—	—	100	—	—
IntelliLayer (full)	—	—	—	—	—	—
A1–A7 ablations	—	—	—	—	—	—

The baseline models B1–B5 do not include a deferral mechanism. Therefore, their automatic-decision coverage is 100% by definition, since every case receives an automated decision.

5.4 Hypotheses and Expected Trade-offs

The proposed experiments aim to investigate four hypotheses. These hypotheses remain unverified until the evaluation is completed.

- **H1 (Decision rule):** Expected-utility-based action selection is expected to achieve higher average utility than threshold-based classifiers when the utility matrix represents asymmetric costs, even when both approaches achieve the same Macro-F1 score.
- **H2 (Deferral):** Referring cases with high uncertainty to human reviewers is expected to reduce the risk associated with automatically handled cases. The full ensemble model is also expected to produce a more favorable risk–coverage curve than a single-network configuration (A2) or a model using Monte Carlo dropout (A3).
- **H3 (Adaptation):** Drift-triggered model updates that incorporate replay and elastic weight consolidation are expected to recover performance more quickly than a system without adaptation (A5) and to retain previously learned knowledge more effectively than ordinary fine-tuning without replay or EWC (A6).
- **H4 (Fusion):** Gated fusion is expected to perform better than simple concatenation (A1), particularly for cases with limited event histories, where static attributes may provide more useful information.

The main anticipated trade-offs involve computational requirements and processing latency. Evaluating multiple ensemble members may increase inference costs approximately in proportion to the ensemble size, while calculating integrated gradients for every decision may introduce additional latency. If gradient-boosted trees (B1) achieve utility comparable to IntelliLayer, as the findings in [8] suggest may occur with purely static data, the primary advantages of the proposed architecture would need to be demonstrated through its

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

uncertainty-based deferral, adaptation capabilities, and auditability rather than predictive accuracy alone. Such a result should be reported transparently.

5.5 Limitations and Threats to Validity

- **Selective labels:** Actual outcomes may be available only for certain actions. For example, the repayment behavior of a rejected loan applicant cannot be directly observed. Training on such selectively observed outcomes can introduce bias, so the evaluation must clearly explain how missing or unobserved outcomes are handled.
- **Utility specification:** The evaluation results depend on the utility matrix supplied by domain experts. Sensitivity analyses across a range of plausible utility values are therefore necessary to determine whether the conclusions remain consistent under different cost assumptions.
- **Uncertainty under distribution shift:** Ensemble uncertainty is an empirical indicator of model disagreement rather than a guarantee of reliability. Certain types of distribution shift may occur without producing a substantial increase in estimated uncertainty.
- **Explanation reliability:** Integrated-gradient attributions depend on the selected baseline and describe how the model responds to its inputs. They should not be interpreted as evidence of causal relationships within the application domain.
- **External validity:** Results obtained from public benchmark streams may not generalize directly to a particular organization's data, outcome delays, operational constraints, or human-review capacity.

6. Conclusion and Future Work

This paper examined the limitations of predictive deep learning models when they are used to support decisions in environments where data patterns change over time. To address these challenges, it introduced IntelliLayer, a five-layer architecture that combines representation learning, uncertainty estimation, utility-based decision-making, adaptive learning, and explainability within a unified framework.

The proposed architecture uses gated fusion to combine static attributes and temporal representations generated by Transformer encoders. A calibrated deep ensemble estimates outcome probabilities and uncertainty, allowing the system to select actions according to expected utility or refer uncertain cases to human reviewers. When ADWIN detects potential concept drift, the adaptation layer prepares model updates using replay and elastic weight consolidation, with a champion–challenger evaluation process controlling whether an updated model is deployed. Integrated-gradient explanations and audit records are generated for individual decisions to support transparency and traceability.

The main contributions include a modular architecture with explicit interfaces connecting confidence estimation, action selection, adaptation, and explanation; a formal online decision-making procedure; a proposed reference implementation stack; and an evaluation protocol focused on decision quality rather than predictive accuracy alone.

However, the expected benefits of IntelliLayer remain hypotheses until the experiments described in Section 5 have been completed. Important limitations include the computational

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

cost of ensemble inference, dependence on a utility matrix defined by domain experts, and potential bias when outcomes are observed only for selected actions.

Future work will focus on four directions:

1. Implementing the proposed architecture and completing Table 4 using public concept-drift benchmarks and a suitable domain-specific dataset.
2. Investigating off-policy evaluation and correction methods to address bias caused by selectively observed outcomes.
3. Developing adaptive deferral policies that account for human-review capacity rather than relying solely on fixed uncertainty thresholds.
4. Exploring computationally efficient uncertainty-estimation techniques that can approach the risk–coverage performance of full deep ensembles while reducing inference costs.

REFERENCES

1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, pp. 436–444, 2015, doi: 10.1038/nature14539.

[2] K. K. Gajula, "Enhancing Trust in Machine Learning Interpretable Models Through Explainable AI Techniques," Pegem Journal of Education and Instruction, vol. 13, no. 4, pp. 909–915, 2023.

[3] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, pp. 529–533, 2015, doi: 10.1038/nature14236.

[4] K. K. Gajula, "Reinforcement Learning with Transparent Policies: An Explainable AI Approach to Adaptive Cyber Security," American Journal of AI Cyber Computing Management, vol. 5, no. 4, pp. 322–328, 2025.

[5] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 11, pp. 7436–7456, 2022, doi: 10.1109/TPAMI.2021.3117837.

[6] K. K. Gajula, "Designing an Ethical Data Science Framework for Responsible and Transparent AI Integration," Emerging Trends in Digital Transformation, vol. 1, no. 1, pp. 74–87, 2025.

[7] X. Wang et al., "Deep reinforcement learning: A survey," IEEE Transactions on Neural Networks and Learning Systems, vol. 35, no. 4, pp. 5064–5078, 2024, doi: 10.1109/TNNLS.2022.3207346.

[8] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[9] K. K. Gajula, "Leveraging Machine Learning Techniques for Earthquake Impact Analysis and Prediction," ZKG International, vol. 9, no. 2, 2024.

[10] A. Vaswani et al., "Attention is all you need," in Advances in Neural Information Processing Systems, vol. 30, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401

[11] X. Wang, B. Zhang, Z. Xu, Y. Qin, and C. L. P. Chen, "An exploration of deep learning for decision making: Methods, applications, and challenges," Expert Systems with Applications, vol. 289, Art. no. 128406, 2025, doi: 10.1016/j.eswa.2025.128406.

[12] K. K. Gajula, "Deep Neural Forecasting Models for Climate Change Analysis Using Multivariate Time-Series Data," Journal of Computational Analysis and Applications, vol. 33, no. 8, pp. 6356–6368, 2024.

[13] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why should I trust you?' Explaining the predictions of any classifier," in Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1135–1144, doi: 10.1145/2939672.2939778.

[14] T. G. Krishna and K. K. Gajula, "Intelligent AI Systems and Advanced Machine Learning: Recent Advances and Real-World Applications," Journal of Intelligent Decision Making and Information Science, vol. 3, no. 7s, pp. 1831–1845, 2026.

[15] M. P. D. Schadd et al., "Reinforcement learning for decision support in defense and security: A systematic review," Annals of Operations Research, 2026. [Online]. Available: https://link.springer.com/article/10.1007/s10479-026-07205-x.

[16] K. K. Gajula, "Advanced Stress Level Forecasting Using Quantile-Optimized Regression Forest Models," International Journal of Engineering Research and Science Technology, vol. 21, no. 4(1), pp. 1–7, Dec. 2025, doi: 10.62643/ijerst.2025.v21.n4(1).pp1-7.

[17] W. Yuan, J. Chen, S. Chen, D. Feng, Z. Hu, P. Li, and W. Zhao, "Transformer in reinforcement learning for decision-making: A survey," Frontiers of Information Technology & Electronic Engineering, vol. 25, no. 6, pp. 763–790, 2024, doi: 10.1631/FITEE.2300548.

[18] K. K. Gajula, "AI-Augmented Web Development: A Competency Framework for the Next Generation Workforce," Journal of Science Engineering Technology and Management Science, 2025.

[19] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in Proc. 34th International Conference on Machine Learning, vol. 70, 2017, pp. 1321–1330. [Online]. Available: https://proceedings.mlr.press/v70/guo17a.html

[20] M. Shahimoon, H. Rasheed, D. R. Rao, K. R. Krishnaiah, K. K. Gajula et al., "Evolving Predictive Models for Customer Churn in Telecommunications," in Proc. 2nd International Conference on New Frontiers in Communication Technologies, 2025.

[21] K. K. Gajula, "XAI-Driven Green Cloud Intelligence for Transparent and Optimized Resource Allocation in Cloud Computing," International Journal of Research in Engineering and Management Sciences, vol. 2, no. 6, pp. 14–29, 2026.

[22] D. Silver et al., "Mastering the game of Go without human knowledge," Nature, vol. 550, pp. 354–359, 2017, doi: 10.1038/nature24270.

[23] K. K. Gajula, "Hierarchical Quantum Learning: A Modular Two-Level Variational Framework for Trainable and Scalable Intelligent Systems," International Journal for Research in Science & Advanced Technologies, vol. 26, no. 2, 2026, doi: 10.65726/ijrsat.2026.V26.I02.10.

This is an open access article under the creative commons license https://creativecommons.org/licenses/by-nc-nd/4.0/

DOI: https://doi.org/10.65713/jeev2i401