Dimensionality Reduction for Complex Event Forecasting

by

Michail Sidiropoulos

Submitted
in partial fulfilment of the requirements for the degree of

Master of Artificial Intelligence

at the

UNIVERSITY OF PIRAEUS

February 2026

Sidiropoulos Michail .....

II-MSc “Artificial Intelligence”

February ,2026

Certified by.....

Elias Alevizos
Senior Researcher
Thesis Supervisor

Certified by.....

Nikos Katzouris
Senior Researcher
Member of Examination Committee

Certified by.....

Georgios Paliouras
Senior Researcher
Member of Examination Committee

Dimensionality Reduction for Complex Event Forecasting

By

Michail Sidiropoulos

Submitted to the II-MSc "Artificial Intelligence" on February, 2026,
in partial fulfillment of the
requirements for the MSc degree

Abstract

Complex Event Recognition and Forecasting (CER/F) has emerged as a pivotal area in artificial intelligence, addressing the need to detect, understand, and predict intricate patterns in dynamic and high-volume data streams. This thesis explores a neuro-symbolic approach to CER/F by integrating dimensionality reduction methods with the Wayeb framework—an automata-based system designed for efficient forecasting. The study focuses on dimensionality reduction as a key strategy to enhance model performance and interpretability, particularly through feature selection methods that identify and retain the most relevant attributes. By reducing the dimensionality of the input data, the alphabet fed into the automaton is simplified, enabling more efficient computations and improved accuracy in CER/F tasks, while preserving the interpretive integrity of symbolic reasoning. To validate the approach, extensive experiments are conducted using synthetic datasets. The impact of dimensionality reduction on recognition and forecasting accuracy, runtime efficiency, and model interpretability is thoroughly evaluated. Results indicate that feature selection significantly improves the scalability of the Wayeb framework and facilitates better generalization in forecasting complex events.

Thesis Supervisor: Elias Alevizos
Title: Senior Researcher at NCSR 'Demokritos'

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Dr Ilias Alevizos for his guidance, support, and valuable feedback throughout the development of this thesis. My deepest thanks go to my parents and sister for their constant encouragement, support and for always believing in me to all of this journey. I am especially grateful to my girlfriend Olga for her patience, understanding, and support during the most demanding periods of this work. Finally, I would like to thank all my friends for their motivation, encouragement and for making this journey more enjoyable, specifically Panagiotis, Giannis, Christos and Fotini.

Contents

List of Figures

1 Introduction

In an era of rapid data generation and increasingly interconnected systems, Complex Event Recognition and Forecasting (CER/F) has become a vital component of artificial intelligence (AI) for real-time decision-making. The demand for real-time processing of large-scale, multidimensional data streams has grown exponentially across numerous domains, such as maritime safety, financial fraud detection, cardiac arrhythmias, and industrial monitoring. By enabling reactive and proactive actions, CER/F systems play an essential role in modern applications requiring timely and accurate decision-making. [1]

A defining characteristic of CER/F is its focus on modeling complex events as compositions of simpler, temporally connected events. By identifying these simpler events and their relationships, CER/F systems can leverage symbolic reasoning to provide interpretable and efficient recognition and forecasting capabilities. This symbolic foundation makes CER/F appealing in domains where explainability and trustworthiness are paramount. However, the inherently high dimensionality and dynamic nature of data streams often introduce computational challenges, including increased runtime and model complexity, as well as reduced scalability. Addressing these challenges requires methods that can effectively reduce data complexity while maintaining the integrity of the information needed for accurate predictions [2].

To efficiently handle real-world data, it is essential to reduce its dimensionality. Dimensionality reduction is the process of converting data from a high-dimensional space into a lower-dimensional one while ensuring that the resulting representation maintains key characteristics of the original data, ideally reflecting its intrinsic dimension [3], [4]. The traditional and the state – of – the – art methods can be generally classified into two main categories, Feature Extraction (FE) and Feature Selection (FS). In general, feature extraction (FE) methods are highly effective in achieving a lower-dimensional space while preserving most of the information. However, they may not be suitable for certain configurations, such as in CER/F models. This is because FE methods transform the data during the dimensionality reduction process, altering the original nature of the features. In contrast, feature selection (FS) methods are better suited for CER/F models, as they reduce dimensionality by eliminating the least informative features while retaining the most relevant ones. This approach ensures that the original structure and interpretability of the data are preserved, which is essential for symbolic reasoning and event recognition tasks [5] [6].

The intricacies of high-dimensional data streams present considerable challenges for Complex Event Recognition and Forecasting (CER/F) systems, especially those utilizing automata-based methods. While automata are adept at symbolic reasoning and temporal pattern detection, their scalability is closely linked to the size of the input alphabet [7], [8]. High-dimensional data can cause this alphabet size to grow exponentially, leading to inefficiencies in both computation and memory consumption. This issue is exacerbated in CER/F models, where real-time processing of dynamic and evolving data streams

demands a careful balance between accuracy and efficiency. Without dimensionality reduction, automata-based CER/F systems encounter constraints such as longer runtimes, limited scalability, and difficulties in managing noise and redundant features. Dimensionality reduction—particularly through feature selection—tackles these challenges by streamlining the input space and retaining the most relevant features, ultimately decreasing the alphabet size and facilitating quicker, more scalable, and interpretable models for recognizing and forecasting complex events in practical contexts [9], [10].

This thesis will utilize the Wayeb framework to generate findings and conduct comparisons, illustrating how dimensionality reduction influences the efficiency and accuracy of automata-based Complex Event Recognition and Forecasting (CER/F) systems. The Wayeb framework is an advanced tool designed for Complex Event Forecasting (CEF) that enables both the recognition and prediction of complex events within real-time data streams. It uses symbolic automata as its core computational mechanism for pattern detection and incorporates Markov chains to predict future events based on those patterns. This integration allows Wayeb to efficiently handle temporal and probabilistic relationships, effectively managing the uncertainties and dynamic characteristics of real-world datasets. By converting event streams into compact symbolic forms, Wayeb improves the scalability and functionality of automata-based forecasting systems, making it especially beneficial for fields such as maritime surveillance, where accurate and timely predictions are crucial.

The remainder of this thesis is organized as follows: Chapter 2 provides a detailed review of the theoretical foundations and existing literature, focusing on Complex Event Recognition and Forecasting (CER/F) and dimensionality reduction techniques. Chapter 3 introduces the methodological framework, detailing how and which feature selection methods are incorporated into the Wayeb framework to enhance its scalability and efficiency. Chapter 4 describes the experimental setup, including the dataset characteristics, preprocessing steps, and evaluation metrics used to assess the proposed approach. In Chapter 5, the results of the experiments are presented and discussed, highlighting the impact of dimensionality reduction on forecasting accuracy, runtime efficiency, and automata complexity. Finally, Chapter 6 concludes the thesis with a summary of findings, limitations, and suggestions for future research directions.

2 Background

This chapter will discuss the essential theoretical concepts related to Complex Event Recognition and Forecasting (CER/F) and dimensionality reduction methods establishing a solid foundation for comprehending the methodologies presented in this thesis.

2.1 Complex Event Recognition – Forecasting (CER/F)

Complex Event Recognition - Forecasting (CER/F), involves identifying patterns within continuously arriving streams of timestamped 'event' data from various sources. These individual instances, termed simple events, are the fundamental occurrences that combine to form complex events. The concept of a complex event is recursive, as it places both temporal and potentially atemporal constraints on its constituent subevents, which can comprise either simple events or other complex events.

On the other hand, complex event forecasting is concerned with predicting future complex events. This differs from classic classification, in that classification categorizes existing simple events as complex, whereas forecasting assesses the likelihood of future events being classified as complex.

CER systems are designed with stringent time-response requirements and must operate efficiently in the rapid analysis of substantial data streams commonly found in Big Data applications [1]. This distinguishes them from conventional database systems, particularly as they do not require the storage of data streams. In various applications, CER facilitates the interpretation of streaming data, enabling systems to respond appropriately and formulate countermeasures. This is what sets CER apart from traditional methods of streaming [11].

Complex Event Recognition (CER) systems take in a stream of time-stamped 'Simple, Derived Events' (SDEs), which are generated through a computational derivation process from other events, such as sensor measurements. By analyzing these SDEs, CER systems can identify complex events (CEs) that form patterns of interest. These patterns establish both temporal and potentially atemporal constraints on the sub-events involved. For example, in maritime surveillance, CER enables the formulation of patterns that lend significance to fused, streaming event tuples, facilitating the real-time detection of suspicious or potentially hazardous situations that could significantly impact environmental safety and navigation at sea [12], [13].

2.2 Automata

Automata serve as a crucial computational model in Complex Event Recognition (CER) because they effectively represent and process patterns of events. In CER systems, automata offer a formal and interpretable method for identifying sequences of events, which facilitates scalable and transparent decision-making. Although the primary focus of this thesis is not on the

mechanics of automata, a brief overview of their theoretical background will be provided to clarify their significance in event recognition and prediction Furthermore the tool which will be used for the experimentation results is based on the very same theorical background analyzed below.

2.2.1 Finite Automata

Finite automata serve as an effective computational model for systems with extremely limited memory, enabling them to perform a wide range of essential tasks despite their simplicity. To formally define finite automata, it is essential to first understand the mechanism with a simple example and its corresponding graphical representation. The following Figure 2. 1 State Diagram of Finite Automaton M1 illustrates a finite automaton called M_1, depicted as a **state diagram**. A state diagram visually represents the states of the automaton and the transitions between them based on input symbols.

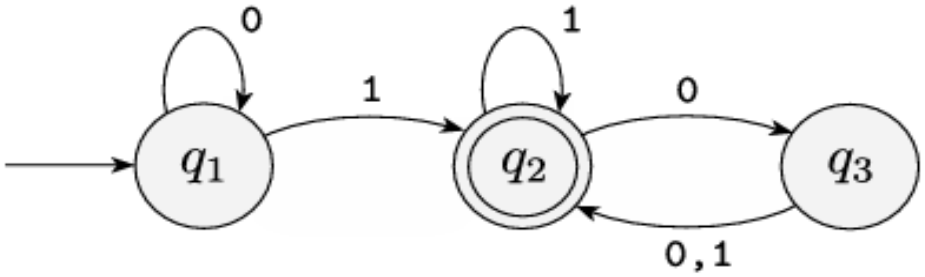

Figure 2. 1 State Diagram of Finite Automaton M1

It has three states, labeled q1, q2, and q3. The start state, q1, is indicated by the arrow pointing at it from nowhere. The accept state, q2, is the one with a double circle. The arrows going from one state to another are called transitions.
When this automaton processes an input string, such as **1100**, it evaluates the string and generates an output, which can either be **accept** or **reject**. For simplicity, we will focus solely on this binary type of output. The computation begins in the automaton's **start state**, where it reads the input symbols. With each symbol read, the automaton transitions from one state to another along the corresponding labeled transition. Upon reading the final symbol, the automaton determines the output. If it reaches an **accept state**, the input is accepted otherwise, it is rejected.
The process of the input string 1100 is:

1. Start in state q_1
2. Read 1, follow transition from q_1 to q_2.
3. Read 1, follow transition from q_2 to q_2.
4. Read 0, follow transition from q_2 to q_3.
5. Read 0, follow transition from q_3 to q_2.

Experimenting with this machine on a variety of input strings reveals that it accepts any string that ends with a 1, as it goes to its accept state q2 and any string that ends with an even number of 0s following the last 1.

To formally define finite automata mathematically, it is essential to first revisit the notation used for strings and languages. In computational theory, a string is

a finite sequence of symbols drawn from a predefined alphabet. The alphabet, typically denoted as Σ, is any nonempty, finite set whose elements are referred to as symbols. A string over Σ is represented as a concatenation of symbols from Σ, including the empty string ε, which contains no symbols.

The language of formal definitions is often rigorous and highly structured, where every detail must be explicitly defined to avoid ambiguity. A finite automaton consists of several key components: a finite set of states, a finite input alphabet (Σ) defining the allowed input symbols, a transition function that specifies state transitions based on input symbols, a start state, and a set of accept states that determine whether an input is accepted or rejected. Mathematically, a finite automaton is represented as a 5-tuple (Q, Σ, δ, q0, F) in Definition 1 [7], [8], [11], [14].

Definition 1. A finite automaton is a 5 – tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the states,
2. Σ is a finite set called the alphabet,
3. $\delta: Q \times \Sigma \to Q$ is the transition function
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

2.3 Symbolic regular expressions to predicate - induced alphabets

This subsection explains how attribute-based pattern descriptions are converted into a deterministic machine that operates over a compact, fixed alphabet. Symbolic patterns, whose transitions are predicates over event attributes, are formalized within a Boolean-algebra framework so they compile into that finite interface and subsequently support probabilistic forecasting. A Boolean algebra of predicates is fixed,

$$\mathcal{A} = (D, \Psi, [\![\cdot]\!], \wedge, \vee, \neg, \top, \bot),$$

where $D$is the (possibly infinite) domain of events,
Ψ a set of unary predicates on D, and
$[\![\psi]\!] \subseteq D$the denotation of $\psi \in \Psi$(the set of events satisfying ψ) [11].
The Boolean connectives are closed in Ψ, and satisfiability/emptiness of any Boolean combination is decidable. Over $\mathcal{A}$, symbolic regular expressions (SRE) are written exactly as in classical regex, but terminals are predicates:

$$R ::= \varepsilon \mid \emptyset \mid \psi \mid (R_1 + R_2) \mid (R_1 \cdot R_2) \mid R^*(\psi \in \Psi).$$

The denotation $L(R) \subseteq D^*$is defined homomorphically:

$$L(\psi) = [\![\psi]\!], \qquad L(R_1 + R_2) = L(R_1) \cup L(R_2),$$
$$L(R_1 \cdot R_2) = \{uv : u \in L(R_1), v \in L(R_2)\},$$

$$L(R^*) = \bigcup_{k \geq 0} L(R)^k .$$

For streaming recognition, a pattern is matched against all suffixes of an

incoming stream. Operationally, this is achieved by recognizing

$$R_s = \top^* \cdot R,$$

where $\top$ is the predicate true on all events
This is equivalent to allowing the automaton to restart at every position.
A Symbolic Finite Automaton (SFA) over $\mathcal{A}$is a tuple

$$M = (\mathcal{A}, Q, q_0, F, \Delta),$$

where $Q$is a finite set of states,
$q_0 \in Q$the start state,
$F \subseteq Q$the set of accepting states, and
$\Delta \subseteq Q \times \Psi \times Q$a finite set of transitions.
A transition $(p, \psi, q) \in \Delta$ is enabled by event $t \in D$iff $t \in [\![\psi]\!]$. The automaton accepts a word $t_1 \ldots t_k \in D^*$if there exists a path

$$q_0 \xrightarrow{\psi_1} q_1 \xrightarrow{\psi_2} \cdots \xrightarrow{\psi_k} q_k \in F\text{with}t_i \in [\![\psi_i]\!] \text{ for all } i.$$

Thus, SREs and SFAs are equivalent description languages for patterns [15] [7] [16].

In classical automata, different labels are disjoint, a single character cannot trigger two distinct outgoing transitions simultaneously. In symbolic automata, different predicates ψ_1, ψ_2 may both evaluate to true on the same event t, so "distinct guards" does not guarantee determinism. An SFA M is deterministic (DSFA) if, for any state q and event $t \in D$, at most one outgoing transition from q is enabled by t. Equivalently, for any two outgoing transitions $(q, \psi_1, q_1), (q, \psi_2, q_2) \in \Delta$ with $q_1 \neq q_2$, their guards must be disjoint:

$$[\![\psi_1 \wedge \psi_2]\!] = \emptyset.$$

Let $\text{Predicates}(M)$be the finite set of guards that appear in M. The determinization first forms the minterms of these predicates—i.e., the set of maximal satisfiable Boolean combinations (atoms) over $\text{Predicates}(M)$—denoted as:

$$N = \text{Minterms}(\text{Predicates}(M)).$$

Each $n \in N$denotes a disjoint, nonempty region of the event domain, and $\{[\![n]\!] : n \in N\}$is a mutually exclusive and exhaustive partition. For example, if $\text{Predicates}(M) = \{\psi_1, \psi_2\}$ then

$$N = \{\psi_1 \wedge \psi_2, \psi_1 \wedge \neg\psi_2, \neg\psi_1 \wedge \psi_2, \psi_1 \wedge \neg\psi_2\}$$

dropping any unsatisfiable combinations. These minterms are then used as the guards of the DSFA [17].

Hence every event $t \in D$ matches exactly one minterm $n = \chi(t)$, the map $\chi : D \to \mathcal{N}$ is total and single-valued. The set $\mathcal{N}$ serves as the predicate-induced alphabet, and its elements are called minterms. If, for a declared attribute a, there

are $k_a$disjoint "buckets" (e.g., specific values or intervals)—including an "other" bucket when the declared values are not exhaustive—then

$$| \mathcal{N} | = \prod_{a \text{ declared}} k_a .$$

Example.
If TransactionResponse $\in \{0, \dots, 5\}$(6 buckets), cnp $\in \{0,1\}$(2 buckets), and amount (continious) is binned into 3 disjoint ranges, then $| \mathcal{N} | = 6 \times 2 \times 3 = 36$. Every event is mapped to one of these 36 minterm symbols.

Given an SFA $M = (\mathcal{A}, Q, q_0, F, \Delta)$and its minterm alphabet $\mathcal{N}$, a deterministic automaton,

$$M^{\det} = (\mathcal{N}, Q^{\det}, S_0, F^{\det}, \Delta^{\det})$$

is constructed over the finite alphabet $\mathcal{N}$ by the standard subset construction, with transitions taken per minterm. States are subsets of Q: $Q^{\det} \subseteq 2^Q$, the start state is $S_0 = \epsilon$ closure($\{q_0\}$) a determinized state $S \subseteq Q$ is accepting if and only if $S \cap F \neq \emptyset$.

The minterm alphabet supplies a finite, mutually exclusive, and exhaustive set of symbols that directly reflect the declared attributes and their granularity. The DSFA provides a deterministic transition system whose evolution on a stream is a function of the current state and the next minterm. These facts enable §2.3 to place a Markov model either on the DSFA's symbols (minterms) or on its states, estimate next-symbol/next-state distributions from data, and derive waiting-time (hitting-time) forecasts for first arrival at an accepting.

2.4 Markov Models for Sequential Data

Automata provide a symbolic and crisp view of patterns over sequences, describing which strings over an alphabet are accepted and which are rejected. In order to forecast how a sequence is likely to evolve, we also need a probabilistic model over these sequences. Markov models offer a simple yet powerful framework for modeling the evolution of a discrete-time stochastic process, such as a stream of event symbols or the sequence of states visited by an automaton.

In the context of Complex Event Forecasting (CEF), Markov models are typically defined over either:

- the symbols consumed by an automaton (e.g., minterms of predicates), or
- the states of the automaton during a run.

In both cases, the key assumption is that the future depends on the past only through a limited amount of recent history [18].

2.4.1 First – Order Markov Chains

Let $(X_t)_{t \geq 0}$be a discrete-time stochastic process taking values in a finite set $\mathcal{S}$(the state space). The process is called a first-order Markov chain if it satisfies the Markov property:

$$\Pr(X_{t+1} = s_{t+1} \mid X_t = s_t, X_{t-1} = s_{t-1}, \dots, X_0 = s_0,) = \Pr(X_{t+1} = s_{t+1} \mid X_t = s_t)$$

for all times $t$and all states $s_0, \dots, s_{t+1} \in \mathcal{S}$. In words, the conditional distribution of the next state depends only on the current state, not on the full history.
A first-order Markov chain is characterized by:

- an **initial distribution** πover $\mathcal{S}$, where $\pi(i) = \Pr(X_0 = i)$, and
- a **transition matrix** $P \in [0,1]^{|\mathcal{S}|\times|\mathcal{S}|}$, where

$$P_{ij} = \Pr(X_{t+1} = j \mid X_t = i).$$

Each row of Psums to 1, since it encodes a probability distribution over the next state.

In automata-based CER/F, one common approach is to let $\mathcal{S}$be the set of states of a deterministic automaton that recognizes a pattern. The sequence of visited states as the automaton processes a stream is then approximated by a Markov chain. This is the basic idea behind Pattern Markov Chains, where the run of the pattern automaton is modeled as a first-order Markov process over its states [18] [19] [20].

2.4.2 Higher-Order Markov Models

The first-order Markov assumption may be too restrictive for many real-world streams, where the next symbol can depend on more than just the immediately previous symbol. To capture longer dependencies, we can use higher-order Markov models.

A process $(X_t)$is an m-th order Markov chain if:

$$\Pr(X_{t+1} = s_{t+1} \mid X_t = s_t, X_{t-1} = s_{t-1}, \dots, X_0 = s_0) = \Pr(X_{t+1} = s_{t+1} \mid X_t = s_t, \dots, X_{t-m+1} = s_{t-m+1}),$$

for all $t \geq m-1$. The next state now depends on the last $m$states, but not on earlier ones.

There are two equivalent perspectives:

1. **Conditional-probability view**
   We estimate conditional probabilities of the form
   $$\Pr(X_{t+1} = s \mid X_t = s_t, \dots, X_{t-m+1} = s_{t-m+1})$$

for all possible contexts $(s_{t-m+1}, \dots, s_t)$.

2. **Expanded state-space view**
   We treat each length-m history $(s_{t-m+1}, \dots, s_t)$as a single state of a first-order chain. The effective state space then has size $|\mathcal{S}|^m$in the worst case.

While higher-order models can represent richer dependencies, they suffer from two major issues:

- **State-space explosion**: the number of parameters (and required training data) grows exponentially with the order m.
- **Data sparsity**: many long histories may rarely or never occur, making probability estimates unreliable.

These problems are especially acute in CER/F, where the alphabet may already be large (e.g., minterms of many predicates), and we would like to capture long-range structure without an exponential blow-up in states [18], [20], [20].

2.4.3 Variable – Order Markov Models

Variable-order Markov models (VMM) address these limitations by allowing the effective order of the model to vary across different contexts. Instead of fixing a global order m, we maintain a set of contexts (suffixes of the past) of variable length, up to some maximum $m_{\max}$.

Formally, as above, let Σ be a finite alphabet (e.g., the set of minterms or DSFA symbols) and let $S_t = s_1 s_2 \dots s_t$ be the sequence observed up to time t. For each time t, the model selects a context c_t, which is a suffix of S_t:

$$c_t \in \{\varepsilon, s_t, s_{t-1}s_t, \dots, s_{t-k+1} \dots s_t\}, k \leq m_{\max},$$

and uses a conditional distribution $\Pr(\cdot \mid c_t)$over the next symbol. The length of the context c_t is not fixed. For some histories it may use only the last symbol, for others it may use a much longer suffix, depends if the data show that the longer context significantly changes the next-symbol distribution.

In practice, variable-order models are usually represented as context trees, where:

- each node corresponds to a context (a suffix of the past),
- edges are labeled by symbols from Σ, and
- each node stores a probability distribution over the next symbol, estimated from training data.

During learning, the tree is grown and pruned according to statistical criteria, so that:

- contexts are expanded (made longer) only when this significantly improves the predictive distribution;
- contexts that do not add information are kept short or pruned entirely.

This adaptive mechanism gives VMMs two key advantages:

1. They can capture long-term dependencies where they exist, by using long contexts.
2. They remain compact and data-efficient, because they avoid modeling unnecessarily long histories in parts of the state space where the process behaves almost without memory occupation.

In the context of this thesis, variable-order Markov models are particularly attractive because they can model how the stream of minterms (or automaton states) tends to evolve, without requiring a prohibitively large number of parameters. In the next section, we focus on a specific and widely used form of variable-order Markov model, the Prediction Suffix Tree (PST), which organizes contexts in a tree structure and is well suited for integration with symbolic automata in Complex Event Forecasting [19].

2.5 Prediction Suffix Trees

In the previous section we discussed Markov models, including variable-order Markov models, which adapt the length of the context used for prediction according to the statistics of the data. Prediction Suffix Trees (PSTs) are a concrete and widely used realization of such VMMs. They represent variable-length contexts in a rooted tree and associate each context with an explicit probability distribution over the next symbol [11].

2.5.1 Definition and Structure

Let Σ be a finite alphabet (for example, the set of minterms produced by determinizing a symbolic automaton). A **Prediction Suffix Tree** Tover Σ is a rooted, directed tree where:

- Each edge is labeled with a symbol $\sigma \in \Sigma$.
- Each node corresponds to a context, i.e., a finite string $s = \sigma_1\sigma_2 \dots \sigma_k$with symbols in Σ. The context associated with a node is obtained by reading edge labels from that node up to the root, so the root corresponds to the empty string ε.
- Each node stores a next-symbol probability distribution $\gamma_s: \Sigma \to [0,1]$, such that

$$\sum_{\sigma \in \Sigma} \gamma_s(\sigma) = 1,$$

where $\gamma_s(\sigma)$is the estimated probability of observing symbol σnext, given that the recent history ends with context s.

The **depth** of a node is the length of its context. The **order** of the PST is the maximum depth $m$of any node. If all leaves are at depth exactly m, the tree represents a full m-th order Markov model: each context of length $m$appears as a path from the root to some leaf, and all shorter contexts are implicitly ignored.

In a general PST, however, different branches may have different depths. Some contexts are kept short (if extending them does not change the predictive distribution), while others are expanded into longer histories when this leads to significantly different next-symbol behavior. This is how PSTs implement a variable-order Markov model, the effective order depends on the particular region of the state space.

2.5.2 Prediction Suffix Trees Learning and Inference

Given a training sequence $S_1..k = t_1, t_2, \dots, t_k$ over Σ and a maximum allowed order m, the goal is to construct a PST that approximates the true next-symbol distribution while keeping the tree compact. Conceptually, the learning procedure has two main phases:

1. Estimating empirical probabilities
   For each candidate context s with $| s | \leq m$ and each symbol $\sigma \in \Sigma$, we estimate:
   a. the empirical probability of seeing context $s$in the training data, and
   b. the conditional probability $\hat{P}(\sigma \mid s)$, i.e., how often σfollows occurrences of s.

These quantities can be computed by counting occurrences of substrings in the training sequence. A convenient data structure for maintaining such counts is the Counter Suffix Tree, where each node represents a string s and stores a counter of how many times s has appeared as a suffix in the stream. Scanning the training sequence once and updating the CST suffices to collect all counts needed for estimating $\hat{P}(\sigma \mid s)$ for $| s | \leq m$.

2. Growing and pruning the Pst

Once empirical probabilities are available, the PST is constructed incrementally:

- The algorithm starts from a trivial tree containing only the root node, representing the empty context ε.
- Candidate contexts are generated by extending existing contexts backwards with symbols from Σ(e.g., from $s$to σs), respecting the maximum depth m.
- For each candidate context s, the algorithm checks whether it should be added as a node.

In this way, the PST expands contexts only when justified by the data. Frequently occurring histories with distinctive future behavior lead to deeper branches, whereas regions of the state space where the process is close to memoryless remain shallow. The resulting tree is a compact, data-driven representation of a variable-order Markov source. Its size depends on the alphabet, the maximum order m, and the thresholds chosen for frequency and distributional change.

Once a PST T has been learned, there are two common ways to use it for forecasting sequences over Σ. A PST can be transformed into a Probabilistic Suffix Automaton, where:

- States correspond to contexts (typically the leaves of the PST, after some additional completion steps),
- Transitions are labeled by symbols $\sigma \in \Sigma$, and
- Each state carries the next-symbol distribution inherited from the corresponding PST node.

This yields an automaton that behaves like a Markov chain of variable order. It is convenient for scenarios where automata-style execution is desired (e.g., when embedding the model into another automaton), but the conversion can increase the number of states significantly compared to the original tree.

Alternatively, one can avoid constructing a PSA and use the PST directly to estimate probabilities of future symbol sequences. Given the current history $S_1..k$, we traverse the PST using (up to) the last m symbols to find the deepest matching context s. The distribution γ_s at the corresponding node gives the next-symbol probabilities Pr $(t_{k+1} = \sigma \mid s)$. By iteratively conditioning on hypothetical future symbols and descending the tree, we can approximate the probability of any finite future sequence. In summary, Prediction Suffix Trees provide a flexible and data-efficient way to model the stochastic evolution of discrete symbol streams. They capture variable-length dependencies in a compact form and can be combined with deterministic automata to support probabilistic reasoning about complex-event patterns [11].

2.6 Complex Event Recognition / Forecasting in Fraud Transactions

Fraudulent behavior in transaction systems is rarely the result of a single isolated action. Instead, it typically manifests as a sequence of suspicious activities that unfold over time. These activities, when viewed individually, may appear harmless or typical, but when combined in specific temporal patterns, they can indicate potential fraud.

2.6.1 Fraud Detection Approaches

Among the various techniques applied in fraud detection, Complex Event

Recognition and Forecasting (CER/F) stands out for its ability to identify meaningful combinations of events within data streams, rather than treating events as isolated occurrences. For example, a fraud pattern might include a sequence of failed login attempts, followed by a successful login from a new location, and then a high-value transaction. While each event alone may not trigger an alert, their temporal dependency and ordered occurrence form a suspicious complex event that could signal fraudulent behavior. CER/F systems are designed to detect such structured patterns in real-time, enabling organizations to take proactive measures before financial damage occurs.

Traditional fraud detection systems often rely on static rule-based approaches, which can be inflexible and slow to adapt to evolving fraud tactics. In contrast, modern approaches leverage machine learning and deep learning techniques to enhance detection capabilities [21]. For instance, Mastercard's AI-powered system, Decision Intelligence, analyzes up to 160 billion transactions annually, assigning risk scores to each transaction in real-time by evaluating vast amounts of data, including purchase history and user behavior. This allows for the identification of potentially fraudulent activities within milliseconds, significantly improving response times and reducing false positives [22].

Complementing these advances, graph-based approaches have gained traction in the field. By modeling transactional data as graphs, where entities like users or merchants are nodes and interactions are edges, these methods can reveal hidden relationships and detect fraud rings that may go unnoticed in traditional analyses. Recent studies have demonstrated the effectiveness of GNNs in financial fraud detection. A comprehensive review highlights that GNNs are adept at capturing complex relational patterns and dynamics within financial networks, significantly outperforming traditional fraud detection methods [23]. Moreover, adaptive GNN models have been proposed to enhance detection performance by learning discriminative representations from transaction data, addressing challenges such as imbalanced datasets and evolving fraud tactics [24].

2.6.2 Fraud Transactions as Complex Events.

Fraudulent activities typically unfold as sequences of interdependent actions rather than as single anomalies, making traditional methods insufficient when facing multi-step or coordinated fraud schemes. Complex Event Recognition and Forecasting (CER/F) is designed to process streams of events over time and recognize composite patterns that emerge across multiple observations.

Fraudulent transactions can be effectively modeled as complex events by treating each individual action in the transaction process as a simple event in a continuous event stream. These simple events may include actions such as account login attempts, payment authorizations, shipping address changes, multiple card rejections, or unusual geolocation accesses. While these actions may appear harmless when analyzed in isolation, their temporal ordering and co-occurrence can reveal sophisticated fraud strategies.

For instance, a complex event pattern might consist of multiple failed login attempts, followed by a successful login from a new device, an address change, and finally a high-value purchase—a sequence that strongly suggests account takeover fraud. Each event is represented as a symbol in the automaton's

alphabet, and the entire sequence defines a complex event pattern represented by a path through the automaton. Temporal constraints (e.g., events occurring within minutes or hours) and user/session correlation are incorporated into the model using parameterized states or context windows. This allows the automaton to track event sequences across sessions and validate the ordering and dependencies between them [4], [25].

While this section has provided an overview of the conceptual modeling of fraud as complex events, a more detailed and formal discussion of how these patterns are specified and recognized using automata-based models will follow in the next sections of this thesis.

2.7 Dimensionality Reduction

High-dimensional data often introduces computational inefficiencies, redundancy, and decreased interpretability, making it challenging for models to process effectively. This section explores various methods for dimensionality reduction, highlighting their role in improving scalability, efficiency, and accuracy in complex event recognition and forecasting (CER/F) systems.

2.7.1 Approaches to Dimensionality Reduction

Dimensionality reduction is the process of reducing the number of features (or dimensions) in a dataset while retaining as much information as possible. In other words, it is a process of transforming high-dimensional data into a lower-dimensional space that still preserves the essence of the original data. This can be done for a variety of reasons, such as to reduce the complexity of a model, to improve the performance of a learning algorithm, or to make it easier to visualize the data. The curse of dimensionality is a common problem in machine learning, where the performance of the model deteriorates as the number of features increases. This is because the complexity of the model increases with the number of features, and it becomes more difficult to find a good solution. In addition, high-dimensional data can also lead to overfitting, where the model fits the training data too closely and does not generalize well to new data. There are two main approaches to dimensionality reduction: feature extraction and feature selection [5].

Feature extraction (FE) refers to the process of transforming raw data into numerical features that can be processed while preserving the information in the original data set. Instead of selecting a subset of existing features, as done in feature selection, FE creates new representations by combining or projecting features into a lower-dimensional space. This transformation can significantly enhance the efficiency and performance of machine learning models by reducing redundancy, noise, and computational complexity. FE is particularly useful in domains where raw data is inherently high-dimensional and complex, such as image processing, natural language processing, and speech recognition. However, in certain configurations, such as those involving symbolic automata, the nature of the features plays a crucial role in event recognition [26]. Since FE methods alter the original feature space by transforming data, they can disrupt the symbolic representations used by automata, making them less interpretable and potentially affecting the accuracy of event pattern matching. This limitation makes feature selection a more suitable alternative in scenarios where preserving the original feature semantics is critical.

Feature selection (FS) is a dimensionality reduction method that removes irrelevant or redundant features while retaining the most informative ones, allowing models to operate efficiently without changing the underlying data structure. Unlike feature extraction, FS works within the original feature space, making it especially appropriate for applications where it is essential to preserve the integrity of individual features, such as in symbolic automata-based systems. In these systems, the input represents symbols that delineate event patterns, and any alterations to their structure could hinder the automaton's ability to accurately recognize and predict events.

Feature selection methods can generally be categorized into three main types: filter, wrapper, and embedded methods. Filter methods evaluate the relevance of each feature based on statistical measures such as correlation, mutual information, or chi-square tests, making them computationally efficient and independent of the model used. However, they do not account for interactions between features. Wrapper methods, in contrast, use a machine learning model to evaluate different feature subsets, selecting the combination that maximizes model performance. While these methods can yield highly optimized feature sets, they are computationally expensive, particularly for large datasets. Embedded methods integrate feature selection directly into the model training process, as seen in techniques like LASSO (L1-regularization) for regression and decision tree-based methods, which inherently rank feature importance. In the context of symbolic automata for CER/F, filter methods are often preferred due to their efficiency and ability to maintain the structure of the feature space, ensuring that the automaton’s input alphabet remains interpretable and computationally manageable [25], [27].

2.7.2 Alphabet Reduction in CER/F

In Complex Event Recognition and Forecasting (CER/F), reducing the input alphabet of an automaton plays a key role in controlling the expressiveness and complexity of the model. As described above at sub-section 2.3, a large alphabet leads to a more fine-grained input space, but it also results in a larger number of transitions and potentially more states, which can significantly increase the size of the automaton and complicate both training and inference. This growth affects not only runtime performance but also the generalization of the automaton, as overly specific symbols can cause fragmentation in the language it recognizes. Alphabet reduction techniques aim to create a more compact and generalizable language, allowing the automaton to focus on meaningful event distinctions rather than noise or minor variations. By simplifying the symbolic representation of events, these methods improve interpretability, reduce the risk of overfitting, and make automata-based CER/F systems more practical for real-time and large-scale applications.

In broader literature, various approaches have been proposed for reducing the input alphabet in automata to address scalability and complexity issues, particularly in learning and model minimization tasks. One prominent method is alphabet abstraction refinement [14], where the alphabet is dynamically adjusted during the learning process to enforce determinism while simplifying the model. This technique allows the system to start with a coarse abstraction and iteratively refine it when nondeterminism is detected, improving the tractability of

automata learning even for large or infinite input domains. Another direction focuses on symbolic automata, which group similar inputs using predicates rather than explicit enumeration of symbols, thus managing infinite or very large alphabets [15]. Additionally, procedure-based size reduction techniques aim to compress automata by identifying repeating subgraphs and reducing redundant transitions, indirectly reducing the effective size of the alphabet handled during pattern matching and recognition [16]. Collectively, these methods emphasize the trade-off between model simplicity and expressiveness, highlighting that careful alphabet reduction can significantly improve automaton efficiency without sacrificing recognition power, which is especially critical in domains like complex event recognition and forecasting (CER/F).

Rather than relying on post-processing techniques or structural compression of the automaton itself, the proposed approach focuses on reducing the size of the input alphabet at the source by selecting only the most relevant features that contribute to the formation of event symbols. This enables the construction of more compact, efficient, and interpretable deterministic automata, well-suited for symbolic pattern recognition and probabilistic forecasting tasks.

2.8 Feature Selection

As previously discussed, an effective approach to alphabet reduction in finite automata involves the elimination of non-relevant features based on their importance scores, as determined by standard feature selection methods. In this section, a theoretical analysis of each feature selection method to be used in the proposed approach will be presented, highlighting their underlying principles and relevance to alphabet reduction in the CER/F context.

2.8.1 Mutual Information Gain

Mutual Information Gain is a supervised feature selection technique that measures the dependence between a feature and a target variable. Rooted in information theory, mutual information quantifies how much knowing the value of a feature reduces the uncertainty about the target. Unlike simpler metrics such as correlation, mutual information can capture non-linear relationships, making it suitable for a wide range of problems. A feature with high mutual information provides significant predictive value, while features with low mutual information are considered less informative. This makes mutual information gain an effective criterion for ranking and selecting features that contribute the most to distinguishing target outcomes, improving both model interpretability and performance. To calculate Mutual information, compute the following formula.

$$I(X;Y) = \sum_{x\in X}\sum_{y\in Y} p(x,y)\log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$

Where,

- p(x,y) is the joint probability of X and Y, respectively
- p(x) and p(y) are the marginal probabilities of X and Y, respectively.

The value of I represents how much knowing the feature X reduces the uncertainty of the target Y. A higher mutual information value indicates stronger dependency and higher relevance of the feature.

In the context of alphabet reduction for automata-based CER/F, Mutual Information Gain is particularly valuable because it focuses on preserving the most informative features that are directly related to the prediction or recognition of event patterns. Since the automaton's alphabet is generated from combinations of feature values, reducing the alphabet based on MI ensures that only the features contributing the most to pattern distinction and forecasting accuracy are retained [28], [29].

2.8.2 Pearson's Correlation Coefficient

Pearson's Correlation Coefficient (PCC) is a classical statistical measure used to evaluate the strength and direction of the linear relationship between two continuous variables. In the context of feature selection, PCC is often applied to quantify how strongly a feature is linearly associated with a target variable. Features with high positive or negative correlation are considered informative, as they provide a predictable relationship with the target. Due to its simplicity and interpretability, PCC is widely used in many data preprocessing pipelines, especially when the relationships between features and targets are expected to be linear and continuous. It is defined as:

$$r_{X,Y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2(y_i - \bar{y})^2}}$$

where,

- x and y are the observed values of X and Y
- $\bar{x}$ and $\bar{y}$ are there respective means

It ranges from -1 to 1, where:

- +1 indicates a perfect positive linear correlation
- -1 indicates a perfect negative linear correlation
- 0 indicates no linear correlation

In the context of alphabet reduction for automata-based CER/F, Pearson's Correlation Coefficient can serve as a preliminary filter to eliminate features that show little or no linear association with the target event patterns. However, it is important to note that PCC only captures linear relationships, potentially overlooking non-linear but informative features [30], [31].

2.8.3 Chi–Squared Test

The Chi-Squared (χ^2) Test is a statistical method used to evaluate the dependency between categorical features and a categorical target variable. It measures how much the observed frequency distribution of feature values deviates from what would be expected if the feature and the target were independent. Higher χ^2 scores indicate a stronger relationship, suggesting that the feature carries significant information about the target. Because it is designed for discrete and categorical data, the χ^2 test is widely used in classification problems to rank and select features that contribute most to distinguishing between different target classes. The Chi–Squared score is computed as:

$$x^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

Where,

- O_i is the observed frequency for category I,
- E_i is the expected frequency for category i under the assumption of independence,
- n is the number of categories or bins.

Its value ranges from 0 to +∞,

- High value means strong dependency between feature and target (relevant feature),
- Low value means feature is independent of the target (irrelevant).

In the context of alphabet reduction for automata-based CER/F, the Chi-Squared (χ^2) Test is particularly appropriate because it is designed to evaluate relationships between categorical variables. As will be discussed later, the dataset selected for the experimental evaluation contains many categorical features, making the χ^2 test a natural choice for identifying which of these features are statistically dependent on the target event patterns [32], [33].

2.8.4 Autoencoder

An autoencoder is a type of artificial neural network designed to learn efficient, compressed representations of data by minimizing the difference between the original input and its reconstructed output. It consists of two main components: the encoder, which compresses the input into a lower-dimensional representation capturing the most relevant features, and the decoder, which reconstructs the input from this compressed form as accurately as possible. For instance, when applied to noisy images, an autoencoder can learn to filter out noise by compressing the image into a cleaner feature space and reconstructing a denoised version of the original. The training objective of an autoencoder is to minimize the reconstruction error, typically using loss functions such as Mean Squared Error (MSE) or Binary Cross-Entropy (BCE), optimized through backpropagation and gradient descent. Autoencoders are widely used in various tasks, including image processing, anomaly detection, noise removal, and feature extraction, where learning meaningful, compact representations of data is essential.

The architecture of an autoencoder consists of three main components: the Encoder, the Bottleneck (Latent Space) and the Decoder Figure 2. 2 Dense

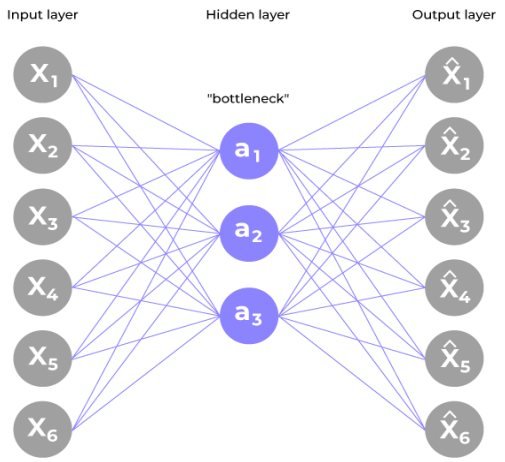

Figure 2. 2 Dense Autoencoder

Autoencoder.

The encoder is the first component of an autoencoder architecture responsible for compressing the input data into a lower-dimensional representation. It starts with the input layer, where the raw data, such as an image or a set of features, is provided to the network. This is followed by one or more hidden layers that apply a series of transformations, progressively reducing the dimensionality of the input while attempting to preserve its most essential characteristics. The output of the encoder is the so-called latent space or latent representation, which serves as a compressed encoding of the original data. This condensed form retains only the most relevant features needed for reconstruction, making it a critical component for tasks such as dimensionality reduction and feature extraction.

The bottleneck, also known as the latent space or code, is the smallest layer in an autoencoder where the data reaches its most compressed form. This layer holds a reduced set of features that captures the most essential information from the original input. The purpose of the bottleneck is to force the network to learn the underlying patterns and structures of the data by removing redundancy and preserving only the most relevant characteristics. This compressed representation serves as the foundation for the decoder to reconstruct the input, making the bottleneck a critical component for efficient data encoding and feature learning.

The decoder is the part of the autoencoder tasked with reconstructing the original input from its compressed representation in the latent space. It typically consists of a series of hidden layers that progressively expand the low-dimensional encoding, attempting to restore the data to its original structure and dimensionality. The process concludes with the output layer, which generates the reconstructed version of the input. The decoder's primary goal is to produce an output that closely matches the original data, minimizing the reconstruction error and validating that the encoder has successfully captured the most meaningful features [34], [35].

Recent research has increasingly positioned autoencoders as valuable tools not only for dimensionality reduction but also for feature selection. Several studies have explored different strategies for integrating feature selection objectives into autoencoder architectures. For instance, Feature Selection Guided Autoencoders (FSAE) introduce learning mechanisms that prioritize important input features during the encoding process, improving both reconstruction quality and interpretability [36]. Similarly, the Selective Deep Autoencoder (SDAE) extends this idea by using a Selective Layer with sparsity constraints, allowing the model to identify a minimal subset of features that preserve the full information content of the data without requiring labels [37]. Other methods, such as AFS (Autoencoder Feature Selector), propose traversing the trained network's strongest connections to select features based on learned importance scores [38]. In parallel, variational and convolutional autoencoders have shown promising results in capturing latent structures for tasks like anomaly detection in multivariate time series, further demonstrating the versatility of autoencoder-based methods in handling both spatial and temporal dependencies [39]. Additionally, Chernikov proposed FRANS, a convolutional network-based method for automatic feature extraction from time series, designed to generate discriminative static features without requiring domain knowledge [40]. Unlike autoencoders primarily focus on reconstruction, FRANS aims to create compact feature representations that improve forecasting accuracy while being computationally efficient.

3 Dimensionality Reduction in CER/F

This chapter addresses the specific research problem of reducing the input alphabet in automata-based Complex Event Recognition and Forecasting (CER/F). Our goal is to improve scalability and runtime efficiency on fraudulent transaction streams by performing feature selection prior to automaton construction. By pruning or coarsening input features that do not materially affect the patterns of interest, we shrink the minterm alphabet that drives determinization, and—consequently—the number of states and transitions in the SDFA that Wayeb uses for recognition and waiting-time forecasting. We evaluate the impact of alphabet reduction on automaton size, throughput, and forecasting quality.

3.1 Problem Space Overview – Problem Definition

As previously discussed, the primary objective of this thesis is to investigate various specification strategies with the overarching goal of optimizing a CER/F system based on probabilistic automata designed for forecasting tasks. The optimization focuses on reducing the size of the alphabet used by the automaton during forecasting, to achieve better computational efficiency, improved scalability, and maintained or enhanced forecasting accuracy by eliminating redundant or non-informative feature combinations. The evaluation of the proposed approach will be conducted using a fraudulent transaction dataset that contains various representative event patterns. A detailed description of the dataset and its characteristics will be provided in the following sections.

As explained above, Complex Event Recognition and Forecasting (CER/F) systems are designed to identify meaningful event patterns in data streams. However, when applied to fraudulent transaction detection, these systems face one major challenge. Transactional data is multidimensional, combining various features like amount, location, device, merchant, and more of which most of them are categorical. Each of these features contributes to the symbolic representation. As the number of features increases the combinatorial growth of the input alphabet makes the automaton larger and computationally demanding at the point of “Alphabet Explosion”. To address this challenge, this thesis proposes the application of dimensionality reduction techniques, and more specifically feature selection methods, with the goal of reducing the alphabet size prior to automaton construction, thereby improving scalability and computational efficiency without compromising recognition accuracy.

3.2 Modeling Fraud Detection as a CER/F Task

Fraudulent transaction detection can be effectively approached as a CER/F problem, where the goal is to identify sequences of interrelated events that collectively indicate suspicious behavior. This section examines how fraud manifests as multi-event patterns within transactional data streams, the challenges posed by the nature of data, and why CER/F provides a suitable framework for detecting and forecasting such complex activities.

3.2.1 Fraud Transaction Domain into CER/F

Fraudulent transaction data can naturally be represented as a continuous stream of events, where each recorded user action, such as login attempts, payment requests, address changes, ID or transaction validations – is treated as a simple event in a larger behavioral sequence. These sequences are not random, they often form structured patterns that reflect legitimate or fraudulent behaviors over time. In the context of fraud, these patterns are typically multi-step processes, making CER/F particularly well-suited.

Moreover, fraud detection operates in a high-throughput, real-time environment, characterized by:

- High Volume: Massive amounts of transaction data generated across multiple channels and systems.
- High Velocity: The need to process and respond to event streams in real time to minimize financial losses.
- High Variety: Transactions that include mixed-type features.

These properties make fraudulent transaction streams a challenging yet representative application domain for CER/F systems, emphasizing the need for efficient, real-time pattern recognition capable of scaling with complex, high-dimensional event data.

3.2.2 Fraud Transaction Dataset

We employ a synthetic (mock) transaction stream explicitly designed to isolate the effect of alphabet (minterm) reduction on SDFA size and forecasting quality while avoiding privacy constraints. The corpus comprises roughly one million timestamped events, partitioned by PAN under a non-overlap policy so that each substream reflects a single card's behavior. Each event includes standard fields (amount, merchant/terminal and card descriptors) from which we derive the event of the target SRE pattern. Additionally, derived change-features are computed to encode short-range dynamics per card (PAN). For each categorical field X(e.g., "Transaction Response", "Merchant Mcc"), events in a PAN-partition are first time-ordered, then a one-step delta is formed as

$$\Delta X_t = X_t - X_{t-1}$$

(with $\Delta X_{t_0} = 0$ for the first observation), this yields "Transaction Response Diff" and "Merchant Mcc Diff" as binary indicators of "change vs. no change." These features serve two purposes:

1) they inject temporal cues that help learning algorithms (e.g., the autoencoder) focus on transitions—such as decline→success or MCC switches—rather than memorize absolute codes,
2) they simplify SRE design and control alphabet size, because predicates like diff = 1 target change events directly, avoiding enumeration of many

absolute values and thereby reducing the number of minterms created during determinization.

To create reliable ground truth for CER/F, the generator plants pattern-consistent sequences at controlled frequencies and injects near-miss sequences and realistic noise. Generator knobs (fraud prevalence, inter-arrival gaps/burstiness, attribute correlations) let us stress-test predicate relevance and observe how selection/coarsening alters the minterm alphabet, the SDFA, and the learned PST.

Preprocessing is minimal and reproducible for each experiment that we running. Temporal and per-PAN splits prevent leakage (no same PAN appears in both train and test). On this controlled stream we quantify, before and after reduction, the alphabet size, SDFA states/transitions, build and waiting-time estimation costs, throughput, and forecasting metrics precision/recall/F1 at fixed threshold, spread and order), attributing changes directly to alphabet reduction rather than uncontrolled dataset drift.

3.2.3 Wayeb

In order to study how feature selection affects complex event recognition and forecasting, we first need a framework that can take a declarative pattern, compile it into an automaton that defines the corresponding event language and states, and then turn that automaton into a probabilistic model so we can estimate when the pattern will be completed. In other words, we need the whole pipeline (pattern → SDFA → PST → forecasting). For this thesis, the framework that provides exactly this CER/F logic is Wayeb.

Wayeb is a complex–event recognition and forecasting engine. It takes a declarative pattern written over streaming events and a historical event stream, and produces, at run time, short forecast intervals that say when a future complex event is likely to complete—together with a probability and a tunable confidence threshold. Internally, Wayeb combines methodologies both from symbolic automata for recognition and variable-order Markov model for forecasting, so it can handle rich, attribute-valued events and long-range dependencies.

In Wayeb, a declarative complex-event pattern must first be compiled into an executable form that can consume the stream and track matches. So, first users specify patterns with symbolic predicates over event attributes (e.g., amount > 100 and country = GR). Wayeb compiles these into a symbolic finite automaton (SFA), then determinizes it. Because symbolic transitions are guarded by predicates (not plain symbols), determinization first builds the minterms—the maximal, mutually exclusive Boolean combinations of the pattern's predicates—and uses those as transition labels. This guarantees that, from any state, at most one outgoing guard is true for a given event, enabling a deterministic SFA (DSFA) suitable for fast streaming recognition. Intuitively, the minterms induce a finite alphabet over an otherwise infinite attribute space. Reducing the predicate set therefore reduces the alphabet and, in turn, the number of minterms and DSFA states. This is exactly where feature selection helps shrink the "language" the automaton must handle.

In earlier work, a deterministic automaton for the pattern $\mathbb{R}$ is converted to a Pattern Markov Chain (PMC) by treating the automaton's run as a 1-step Markov process. For higher-order dependencies one builds an m-unambiguous

automaton that “remembers” the last *m* - *unambiguous* symbols, then estimates the chain’s transition matrix from data. Forecasting then comes from waiting-time distributions, for each non-final state *q*, computes the distribution of the number of steps until any final state is first reached (a standard absorbing-chain first-passage computation). This yields, for each state, a completion-probability curve over “steps ahead” from which Wayeb selects the shortest interval whose mass exceeds a user confidence threshold θ. The approach is principled but suffers when *m* grows (state space and data demands escalate quickly).

To capture longer-range structure without the blow-up of a full order model, Wayeb learns a Prediction Suffix Tree (PST), a variable-order Markov model whose nodes are “contexts” of variable length (suffixes of the recent event history) and whose edges store the next-symbol conditional probabilities. The PST only expands contexts when the data justify it, so it can represent long dependencies selectively. The learned PST can be turned into a Probabilistic Suffix Automaton (PSA) or used directly to drive forecasting. In both views, you get a Markovian next-event model that is as deep as needed where it matters, and shallow elsewhere.

At run time, incoming events are classified by the DSFA via the minterm that matches the event (recognition) and simultaneously appended to the current PST context (modeling). This coupling lets Wayeb ask “Given the current DSFA state how close we are to the pattern and the PST - based next event probabilities how the stream tends to evolve from this context, what is the distribution of the number of future events until any final DSFA state is first hit?” That distribution is the waiting-time distribution for the current state/context. It is updated implicitly as the state and the PST context evolve. Wayeb then extracts a concise forecast interval [start,end] whose probability mass exceeds the threshold θ, preferring the smallest such interval.

Wayeb supports both regression-style interval forecasts (pick the tightest [s,e] with mass ≥ θ) and classification-style windows (e.g., positive if completion occurs within the next k events). In all cases, the horizon caps how far ahead the model considers, the max spread caps interval length and the confidence threshold θ trades off precision and recall. The higher θ yields fewer—but more confident—forecasts, typically later, the lower yields more—and earlier—forecasts at the risk of false positives [41].

Why symbolic + PST matters for your thesis (feature selection). Because determinization uses minterms, the number of DSFA states the size of the state space over which waiting-time distributions are computed and the diversity of contexts the PST must explain—grows with the number and granularity of predicates. Feature selection (and predicate coarsening) reduces the predicate set and the partition it induces on the data, shrinking the minterm alphabet and the DSFA. This has two downstream benefits:

1) Learnability—the PST concentrates data on fewer, more meaningful contexts, improving probability estimates
2) Efficiency—smaller automata and trees cut memory and speed up recognition and forecasting.

All of this keeps the method formal, compositional, and fast enough for real streams, while allowing high-order dependencies where they matter

3.3 Feature Selection for Alphabet Reduction

This section presents the feature-selection stage that we introduce in order to reduce the size of the automaton's input alphabet. It also explains the different feature-selection methods that considered and how their outputs are combined into a single aggregated importance score that drives the final predicate set.

3.3.1 Motivation and Position in the Pipeline

In Wayeb, the number and granularity of symbolic predicates that appear in the SRE and in the accompanying declarations have a cascading effect on the size of the constructed automaton. Furthermore, declarations is the place where the alphabet of the forecaster is designed. It specifies the unary predicates (value bins and categorical splits) that partition the event space before determinization, along with constraint. Conceptually, it augments the predicates implicit in the pattern without changing the pattern's language, instead, it controls how each event is mapped to a minterm symbol and therefore how many symbols the DSFA and the Markov model must handle. For example, let's say that the following pattern needs to be forecasted in a respective dataset.

"Amount = 100 followed by Amount = 200 followed by Amount = 300".
Formally, if the only predicates in scope are the three equalities ψ_{100}: EQ(amount, 100), ψ_{200}: EQ(amount, 200), ψ_{300}: EQ(amount, 300),
then without any declarations Wayeb treats these guards as black-box predicates and does not assume they are disjoint. Mintermization therefore enumerates all $2^3 = 8$ truth assignments over $\{\psi_{100}, \psi_{200}, \psi_{300})$ and keeps them (including overlap atoms like $\psi_{100} \wedge \psi_{200}$), yielding $|N| = 8$. With a mutual-exclusivity declaration $+(\text{EQ(amount, 100)}, \text{EQ(amount, 200)}, \text{EQ(amount, 300)})$ the overlap atoms are ruled out and only the disjoint cases survive, together with the complement("other"):

$\psi_{100} \wedge \neg\psi_{200} \wedge \neg\psi_{300}$,
$\neg\psi_{100} \wedge \psi_{200} \wedge \neg\psi_{300}$,
$\neg\psi_{100} \wedge \neg\psi_{200} \wedge \psi_{300}$,
$\neg\psi_{100} \wedge \neg\psi_{200} \wedge \neg\psi_{300}$,
so $|N| = 4$.

Each additional predicate can participate in the formation of minterms, and minterms are exactly the symbolic input symbols on which the deterministic SFA is built, as a result, a richer predicate set leads to a larger alphabet, and a larger alphabet leads to more states and transitions after determinization. Following the above example, let's say in the same configuration we add declarations such as:

EQ(CNP,0), EQ(CNP,1) (mutually exclusive)
Because Wayeb does not assume coverage unless explicitly enforced (or guaranteed by the domain), the cnp split yields three buckets after mintermization: cnp=0, cnp=1, and the complement "other" (neither 0 nor 1). The joint alphabet is the Cartesian product of the per-attribute buckets, so the new minterm count is $4 \times 3 = 12$. If, in addition to exclusivity, coverage for cnp is asserted (or if the system is told that cnp is strictly binary), the cnp side collapses to two buckets and the alphabet becomes $4 \times 2 = 8$.

At the same time, declarations are not "free to remove": every declaration is

also a piece of information about the event that can help the model distinguish between similar paths and, later, produce more accurate waiting-time forecasts. Choosing the number of declarations is therefore a trade-off, too many and the minterm alphabet explodes, making recognition and forecasting heavier. Too few and the model may lose signal that would have improved the quality and earliness of its predictions.

Fraud-oriented event streams, even when synthetic, typically expose a wide set of attributes such as amount, acquirer or card country, terminal descriptors and response codes. If all of these attributes are turned into separate symbolic predicates and fed unchanged into the SRE/declarations, they all become candidates for minterm construction. This leads to a situation where many predicates are present in the specification but only a subset of them actually contributes to the progress of the automaton toward a match. Predicates that are very sparse in the stream, that are highly correlated with others, or that do not affect the reachable transitions of the SDFA inflate the symbolic alphabet without offering additional discriminative power. The result is a heavier determinization phase and, consequently, a larger SDFA than what is strictly needed for the target fraud pattern.

For this reason the selection step must be placed before minterm construction and determinization. If predicates are pruned or coarsened after minterms have already been generated, the automaton has already paid the cost of the larger alphabet and the state space cannot be reduced effectively. By contrast, inserting feature/predicate selection directly after the SRE and declarations means that only the retained predicates will participate in minterm formation, so the alphabet, the SDFA, and later the PST and waiting-time estimation will all be built on a smaller, task-relevant input space. The resulting pipeline is presented at Figure 3. 1.

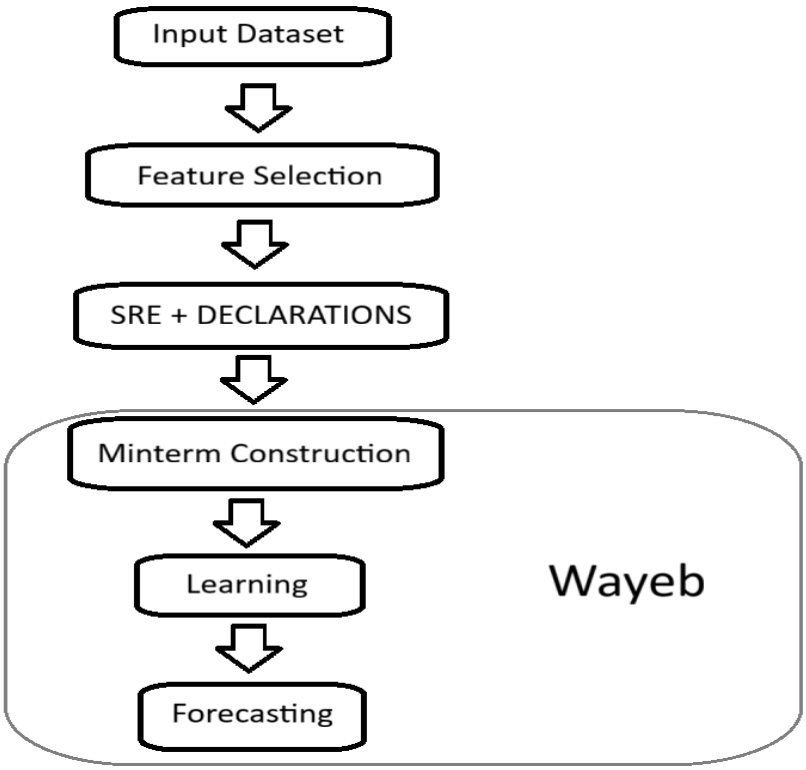

Figure 3. 1 Experiment Configuration Pipeline

The objective of this stage is to reduce the alphabet and the automaton size while preserving the semantics of the target pattern and maintaining forecasting metrics (precision, recall, earliness) within predefined bounds.

3.3.2 Predicate Generation from Selected Features

After feature selection, the remaining attributes must be turned into concrete SRE declarations so that Wayeb can consume them. This mapping is straightforward but important. Categorical attributes (e.g. terminal, country, response code) are expressed as equality predicates (EQ(field, value)), while numeric attributes (e.g. amount) are first discretized into a small number of bins and then each bin is expressed as a separate predicate (e.g. amount ∈ [0,100), amount ∈ [100,500)). Derived attributes such as TransactionResponseDiff are also kept as equality predicates because they directly encode the progression pattern we want to detect. In this way, every selected feature is made visible to Wayeb through one or more symbolic conditions. When we define these predicates, we keep them mutually exclusive wherever possible (e.g. non-overlapping amount ranges), so that each event satisfies at most one predicate per attribute. This makes the later minterm construction simpler and avoids creating extra combinations that do not add information.

This is also the point in the pipeline where we effectively control the final alphabet size. Even if an attribute has been selected as informative, creating too many predicates for it (for example, very fine-grained amount intervals or one predicate per rare categorical value) will still cause minterm blow-up during determinization. Conversely, using coarser bins for numeric fields and grouping infrequent categorical values into an “other” category keeps the number of predicates per feature low and leads to a smaller symbolic alphabet. Therefore, predicate generation from selected features is not just a mechanical step, but the place where we balance informativeness and alphabet compactness.

3.3.3 Evaluation of the Reduced Alphabet

The evaluation is designed as an incremental study rather than a single baseline–vs–reduced comparison. We begin from the extreme case where no feature-derived declarations are provided to Wayeb, so minterm construction is minimal and the automaton is very small, but the model has almost no information about the structure of the events. Then, following the global feature-importance ranking obtained in Section 3.3.2, we add declarations step by step. For every step we regenerate the declarations file, rerun the Wayeb pipeline, and measure the effect of the additional predicates. This setup matches the main hypothesis of the thesis: there exists a region in which adding informative predicates improves recognition/forecasting, and beyond that region additional predicates mostly increase automaton complexity.

For each declaration budget we collect two groups of measurements. The first group is structural:

- induced alphabet size $|\Sigma|$,
- number of SDFA states,
- number of transitions,
- determinization/waiting-time estimation time.

These directly show how much the automaton grows as we expose more attributes. The second group is operational: the CER/F metrics that Wayeb already reports (precision, recall, F1, specificity, etc) under the same input hyperparameters, which will be discussed further in the following section, together with throughput (events/sec). Plotting these quantities against the number of declarations makes the trade-off visible, initially the metrics rise, because the model receives genuinely useful information, but after a certain point. The structural cost keeps increasing while the forecasting quality plateaus or even decreases (e.g. due to over-fragmentation of the alphabet). The “knee” of this curve is the declaration budget we consider most appropriate for this dataset and pattern.

4 Experiment Setup

This section describes the experimental setup used to assess the impact of performing feature selection before running Wayeb on alphabet size, automaton complexity, and forecasting performance. We first outline the environment, dataset preparation, and feature-selection stage, and then detail how Wayeb was configured and executed for a sequence of increasing declaration budgets.

4.1 Overview

The experiments in this chapter aim to show, in a controlled way, what happens to a CER/F pipeline when we reduce the number of attributes that are exposed to the automaton through declarations. In earlier sections we argued that declarations determine the predicates, predicates determine the minterms, and minterms determine the SDFA size, so reducing the number of declarations should shrink the automaton and potentially speed up forecasting. Here we turn that into something measurable: we run exactly the same pattern, on the same stream, with exactly the same Wayeb parameters, and only change how many features we allow to become declarations. By doing this incrementally we can observe the point where adding more information stops helping and starts to make the automaton heavier.

All experiments are carried out on the synthetic fraud transaction dataset described in Section 3, which already contains the fields we need and is large enough (~1M events) to expose determinization and forecasting costs. Using a synthetic but fixed dataset is important here because we want differences in the results to be attributable to our intervention (feature selection → different declarations) and not to data drift or changing class balance. The stream is processed in the same way in every run (partitioned by PAN, non-overlap policy), so the only variable is the size and content of the declaration file. The rest of Section 4 details how we built these runs environment, feature-selection stage, generation of declarations, and the exact Wayeb configuration.

4.2 Environment and Tools

The experiments were conducted on a single workstation. The system was equipped with Windows 11 pro operating system with the following specifications:

- AMD Ryzen 5600U 2.3Ghz
- 16Gb Memory Ram
- Nvidia GeForce MX450 with 4Gb of vram

All executions of the CER/F pipeline used the same machine so that differences in build time, forecasting time, or throughput can be attributed to the different declaration sets rather than to hardware variability.

Wayeb was executed through its Scala/CLI interface using the same JDK version in all experiments. The CLI was invoked from the project's root directory, and the pattern file, declarations file, input stream, and results directory were passed as arguments, as shown later in this section. Where necessary, the JVM was started with increased heap space to accommodate determinization and PST learning, since both stages are sensitive to the number of minterms produced. No other performance tuning was applied, because the goal was to evaluate the relative effect of alphabet reduction under realistic, repeatable settings.

For the feature-selection stage, which is executed before running Wayeb, we used a separate Python environment. The code was developed in Python 3.8.6 and relied on common data-science libraries:

- Pandas 2.0.3, for data loading and preprocessing,
- NumPy 1.24.4, for numerical operations,
- Scikit-Learn 1.3.2, for the feature-selection methods, scalers and other,
- Matplotlib 3.7.4, for plotting the results,
- PyTorch 2.3.0, to define and train a dense autoencoder in order to obtain reconstruction-based importance scores.

All experiments were run in a reproducible environment using fixed library versions, and the outputs of this Python stage were simply the ranked feature lists that were then translated into Wayeb declaration files.

4.3 Fraud Transaction Dataset Preparation

Since the goal of the experiments is to measure the effect of changing the declaration set, the dataset itself had to be made consistent. The fact that the dataset is synthetic through a generator make it full configurable to generate datasets appropriate for the specific problem. All three datasets generated consists of 1 million events both genuine and fraudulent with percentage of 0.1 % fraudulent transactions. Each fraudulent transaction is characterized by the same card PAN number which is a unique number identifier to keep the nature of the problem accurate. In the following bullet list, the remaining features are presented together with their real-world meaning, value type, and typical range.

- **timestamp**: the exact time (in milliseconds) when the transaction was made/processed.
- **TransactionId**: unique identifier of this transaction in the system. Used to trace/log. Eg. "13ce1d7c0d954170b7a91dd3c40c9d66".
- **CNP**: "card not present" flag (e.g. e-commerce, phone order). 1 = card not physically present, typically higher fraud risk; 0 = card present.
- **Amount**: monetary value of the transaction (247.4). Key for spending patterns, limits, and anomaly checks. Float in range (0.0, 1000.0).
- **Pan**: the (tokenized/hashed) Primary Account Number of the card. Identifies the card/customer used to partition the stream. Eg. "c8fadf19edab405d803ecd4f93948b3d"
- **CardExpDate**: card expiration date (here in yyyymm or similar packed form). Basic card attribute, sometimes used for validity checks. Eg. "200902"

- **CardCountry**: country that issued the card. Useful for geo-consistency checks (card from country X used in country Y). Integer, range (0,5).
- **CardFamily**: product family of the card (e.g. classic, gold, debit/credit grouping defined by the issuer). Encodes segment/risk. Integer, range (0,5).
- **CardType**: type of card (e.g. credit, debit, prepaid). Different types have different normal usage patterns. Integer, range (0,5).
- **CardTech**: technology of the card (magstripe, chip, contactless/NFC). Ties to what the terminal should see. Integer, range (0,5).
- **AcquirerCountry**: country of the acquiring bank/terminal. Used to detect cross-border or unusual locations. Integer, range (0,5).
- **MerchantMcc**: merchant category code. Tells what kind of business it is (retail, fuel, online services). Important for behavioral profiling. Integer, range (0,5).
- **TerminalBrand**: brand/vendor of the POS/ATM terminal. Sometimes used for device-level risk or configuration. Integer, range (0,5).
- **TerminalId**: identifier of the specific terminal. Lets you see repeated transactions on the same device. Integer, range (0,5).
- **TerminalType**: type of terminal (POS, ATM, unattended, mobile POS). Different risk per type. Integer, range (0,5).
- **TerminalEmv**: EMV capability / mode of the terminal (chip-enabled, fallback, etc.). Non-EMV or fallback can be riskier. Integer, range (0,5).
- **TransactionResponse**: outcome code of the authorization (approved, declined, referred, etc., here coded as 5). This is what your pattern later tracks. Integer, range (0,5).
- **CardAuth**: result or method of card-level authentication (e.g. chip/PIN OK). Helps distinguish why a transaction was accepted. Integer, range (0,5).
- **TerminalAuth**: authentication/status at terminal side (terminal's view of auth). Can show terminal misconfigurations or suspicious flow. Integer, range (0,5).
- **ClientAuth**: customer-side authentication result (e.g. PIN, 3DS, OTP). Stronger auth → lower fraud likelihood. Integer, range (0,5).
- **CardBand**: issuer-specific banding/grouping of cards (sometimes maps to product/risk tiers). Integer, range (0,5).
- **CvvValidation**: result of CVV/CVC check. Failed or absent CVV in card-not-present is a risk signal. Integer, range (0,5).
- **TmpCardPan**: temporary/masked/card-on-file identifier used internally (e.g. tokenization). Lets the system link related transactions safely. Eg. "0c500386c83f43bdaa2b1e3c566bb260"
- **TmpCardExpDate**: expiration date for the temporary/masked card token. Mirrors the real card's expiry. Eg. "200902"
- **TransactionType**: nature of the transaction (purchase, refund, cash advance, etc). Behavior depends on type. Integer, range (0,5).
- **AuthType**: which authorization flow was used (online, offline, 3DS, issuer-specific). Different flows have different guarantees. Integer, range (0,5).
- **isFraud**: ground-truth label for fraud (0 = not fraud). Used for training/evaluation. Integer, range (0,1).("Target")

To obtain statistically meaningful results, and since the generator was easily configurable, we produced three separate datasets. Each dataset consists of genuine transactions and embeds a single fraud pattern that appears multiple times. The reason behind this is to illustrate different scenarios and prove the robustness of the pipeline in different situations. Since the generator is fully configurable some of the patterns that we chose to include are not possible fraudulent in a real-world scenarios but instead are represented as fraudulent because there are out of the genuine transaction's distribution. With that we achieve an easier interpretable solution since we can configure also how many actual features are corelated with the fraud pattern itself. The latter is a valid approach to simulate such an action since, a pattern can only be captured and been known when it happens and with the easiest observable feature. So, the patterns that implemented contain one or two of the features and the others are added to the declaration which will be discussed later.

Also, all of the events were ordered by timestamp, partitioned by PAN so that each substream represents a single card, and enriched with the derived field that is used in the target pattern. We also applied light discretization to numeric attributes and consolidated rare categorical values, because these transformations directly influence how many predicates will be generated later. In two of the datasets, we augmented the first two patterns with a derived difference attribute per PAN, defined as:

$$\Delta x_t = x_t - x_{t-1}$$

e.g., MerchantMccDiff, TransactionResponseDiff, so that each event carries first-order temporal change information. This lightweight temporal signal is intended to inform the feature-selection stage—discussed in detail later—so the filters can prioritize attributes that are predictive of short-range progression toward pattern completion, and to provide an additional interpretable attribute that improves the explainability of the resulting forecasting patterns.

The first pattern "Merchant Mcc Diff" consists of four consecutive events characterized by low amount transactions within ten-minute window time involving multiple countries and merchant categories with the same Card Pan number. The pattern that stayed the same was:

- [Amount < 2] -> [Amount < 2 and MerchantMccDiff = 1] -> [Amount < 2 and MerchantMccDiff = 1] -> [Amount < 2 and MerchantMccDiff = 1]

The above is a formal representation of the pattern which actually means that we seek for an event with amount lower than value 2.0 followed by three more events with amount lower than value 2.0 **and** attribute MerchantMccDiff equals to 1.0. This pattern was chosen for plausibility it models a stolen-card scenario in which the perpetrator attempts multiple low-value purchases across different merchants to avoid immediate detection by the cardholder or issuer. While this constitutes the core of the pattern, we also incorporate additional attributes that contribute to the overall event representation. The extra features are:

1. CNP, which has always the value 0.0,
2. Acquirer Country, which in the first occurrence has a random number and in the next ones we add the number 1.0.

The second pattern “Decline Before Success” comprises six consecutive events related with the attribute “Transaction Response”, five consecutive transaction declines followed by a final successful transactions within five - minute window time of the same Card Pan number. The pattern that stayed the same here is:

- [TransactionRespDiff = 0] -> [TransactionRespDiff = 1] -> [TransactionRespDiff = 1] -> [TransactionRespDiff = 1] -> [TransactionRespDiff = 1] -> [TransactionRespDiff = 1]

Formally, this specification seeks six card-present events for the same PAN within a five-minute window such that the first five are declines TransactionResponse $\in$ {1,2,3,4,5} and the sixth is an approval (TransactionResponse = 0). We encode this progression via the derived attribute TransactionResponseDiff, using the sequence [0, 1, 1, 1, 1, 1] to denote an initial reference event (no prior response; diff = 0) followed by five changes in response state (diff = 1). The scenario is plausible for “decline-before-success” fraud, a perpetrator repeatedly attempts purchases, often adjusting amounts, until an approval slips through. This pattern simulates a real-life situation to which someone tries to brute force credentials and in the final transaction gets it. While this constitutes the core of the pattern, we also incorporate additional attributes that contribute to the overall event representation. The extra features are:

1. Amount, fixed values {120, 110, 100, 90, 80, 150},
2. Terminal Emv, fixed same value along the fraud events,
3. Terminal Id, fixed same value along the fraud events.

The third pattern “Card Limit” comprises six consecutive events characterized by fixed increasing transaction amounts, with five consecutive approvals followed by a final declined transaction within a four-minute window of the same Card Pan number. The pattern that stayed the same here is:

- [amount = 1000] -> [amount = 1100] -> [amount = 1200] -> [amount = 1300] -> [amount = 1400] -> [amount = 1600]

Formally, this specification seeks six card-present events for the same PAN within a four-minute window such that the first five are approvals with monotonically increasing followed by a sixth transaction at a higher amount equal to 1600 that is declined TransactionResponse equals to zero. This pattern models a limit-probing scenario, the perpetrator issues a series of increasingly costly transactions to discover the effective card limit or issuer threshold, culminating in a decline once the limit is exceeded. The extra features are:

1. Transaction Response, fixed values {0, 0, 0, 0, 0, 1}
2. Merchant Mcc, fixed same value along the fraud events,
3. Terminal Id, fixed same value along the fraud events,
4. Terminal Emv, fixed same value along the fraud events.

4.4 Feature – Selection Stage.

In this section, we present the configuration of the feature-selection methods employed and the aggregation scheme that determines the set of features used in the final forecasting model.

4.4.1 Classic Machine Learning Methods

This subsection concisely documents the classical machine-learning filters used for feature selection. Since the theory and implementation details were presented earlier, we focus here on hyperparameters, preprocessing choices, and selection rationale for each method, to ensure reproducibility and to justify why each technique is included in the aggregation.

Before jumping into the different methods used, lets discuss about how we prepared the data for the statistical methods. Since there are some features that are more "confusing" to the algorithms we dropped them. We dropped the following features:

1. Timestamp
2. Transaction Id
3. Card Pan Number
4. Tmp Card Pan
5. Tmp Card Expiration Date
6. Card Expiration Date

Also for some methods which are unsupervised we also dropped the target which is "isFraud" and "FraudType".

Firstly, we chose to use Pearson's Correlation Coefficient (PCC) as a simple, interpretable filter to assess the linear relationship between each feature and the fraud label. After dropping the purely identifying attributes (timestamp, transaction id, card PAN and its temporary variants, and expiration dates), we treated the isFraud field as a binary numeric target and computed the Pearson correlation between every remaining feature and this target on the training split. Since the majority of attributes in the dataset are either numerical or integer-encoded categorical variables with a small value range, no additional scaling was required, as PCC is inherently scale-invariant. In practice, we worked with the absolute value of the coefficient |r|, so that both positively and negatively correlated features are considered equally informative. Features whose correlation with isFraud was very close to zero were marked as weak candidates, while features with higher |r| were kept as potentially useful predictors. Besides the feature–target correlation, PCC was also inspected for pairs of features, in order to spot attributes that are almost linearly dependent and therefore likely to convey very similar information in the later forecasting stage. The following hyperparameters are used from the library sklearn.feature_selection r_regression [31]:

- force_finite = True, whether or not to force the Pearson's R correlation to be finite. In the particular case where some features in X or the target y are constant, the Pearson's R correlation is not defined. When force_finite = False, a correlation of np.nan is returned to acknowledge this case. When force_finite = True, this value will be forced to a minimal correlation of 0.0.

- center = True, whether or not to center the data matrix X and the target vector y. By default, X and y will be centered.

In parallel with PCC, we applied Mutual Information (MI) Gain as a supervised criterion to quantify how informative each feature is with respect to the fraud label. Using the same cleaned set of attributes, after dropping irrelevant features, we treated "isFraud" as the target variable and estimated the mutual information between each remaining feature and this label. Since most features are discrete, integer-encoded variables with a small range of values, MI is a natural choice, as it can capture both linear and non-linear dependencies without requiring additional scaling. In practice, we used a standard implementation of mutual-information–based feature selection, which returns a non-negative score per feature, interpreted as the expected reduction in uncertainty about the target when that feature is observed. Features with MI values close to zero were considered weak candidates, while features with larger MI scores were regarded as more relevant for distinguishing fraudulent from genuine transactions and therefore more promising for inclusion in the forecasting pipeline. The following hyperparameters are used from the library sklearn.feature_selection mutual_info_classif [29]:

- discrete_features = auto, if bool, then determines whether to consider all features discrete or continuous. If array, then it should be either a boolean mask with shape (n_features,) or array with indices of discrete features. If 'auto', it is assigned to False for dense X and to True for sparse X
- n_neighbors = 3, number of neighbors to use for MI estimation for continuous variables. Higher values reduce variance of the estimation, but could introduce a bias [42].
- n_jobsint, default = None, the number of jobs to use for computing the mutual information. The parallelization is done on the columns of X. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors.
- random_state = None, determines random number generation for adding small noise to continuous variables in order to remove repeated values. Pass an int for reproducible results across multiple function calls.

Lastly, the final classical machine learning feature selection method we employed is the **chi-squared (χ^2) statistic**. In this setting, χ^2 is used as a filter-based criterion to measure the degree of dependence between each (discrete) feature and the binary fraud label. Working again on the cleaned set of attributes, we exploited the fact that most of the features are already non-negative, integer-encoded categorical variables, which satisfies the basic assumptions of the χ^2 test and removes the need for additional preprocessing. For each feature, we constructed a contingency table between its possible values and the two outcomes of the isFraud label, and then computed the corresponding χ^2 score. Intuitively, features whose value distributions differ significantly between fraudulent and genuine transactions receive a high χ^2 value, while features whose distributions are similar across the two classes obtain scores close to zero. As with the other filter methods, we treated larger χ^2 scores as evidence that a feature is more discriminative and therefore more suitable to be kept in the reduced feature set that will later be passed to the forecasting pipeline. The discussed method is not configurable with any hyperparameters the library from sklearn feature_selection chi2 [33].

4.4.2 Dense Auto – Encoder as Feature Selection Method.

In addition to the classical filter methods, we implemented a dense autoencoder as a neural, embedded feature selection mechanism, but first we briefly discussed the datasets used for this method. Since temporal information has not been explicitly incorporated into any of the feature selection procedures described so far and given that it is a core aspect of forecasting, we now describe the configuration of the dataset to reintroduce and exploit the temporal dimension. In order to add this dependency to the data, each original event (transaction) is no longer treated in isolation instead, we construct sliding windows of fixed length over the chronologically ordered stream. Concretely, for a window size, every new instance fed to the autoencoder is formed by concatenating the feature vectors of n consecutive events, $(x_t, x_{t+1}, \dots, x_{t+window\_size})$,into a single, higher-dimensional vector. Given an initial sequence of N events with F features each, this transformation yields N−w+1 overlapping windows of dimensionality $F \times w$. Before windowing, the numerical attributes are scaled (with a separate treatment for the amount field), and two separate JSON datasets are prepared for training and validation. In this way, each input sample to the autoencoder encodes not only the state of one transaction, but also its short-term temporal context, allowing the network to learn patterns that span multiple consecutive events, which is more consistent with the downstream forecasting task.

Using the same preprocessed feature set (after removing timestamps, identifiers and PAN-related attributes and applying separate scaling to the amount field), we then trained a fully connected autoencoder to reconstruct each sliding-window vector from a lower-dimensional latent representation. The network consists of an encoder that maps the flattened $F \times w$ input space to a bottleneck layer and a decoder that expands this latent code back to the original dimensionality. Training is performed in an unsupervised way on the training split by minimizing the mean squared reconstruction error between inputs and outputs with the Adam optimizer. Once the model converges, we use the weights of the first linear layer in the encoder, which directly connect each input dimension (i.e., a specific feature at a specific time position within the window) to the first hidden layer—as a proxy for how strongly the model relies on that input. Because the flattened input vector has size $F \times w$, each original feature appears w times, once for each temporal position in the sliding window. Concretely, we take the absolute values of these weights and average them across hidden units to obtain one global importance score per input dimension, then reshape the resulting vector into $F \times w$ matrix and average over the w time positions to derive a single score per original feature. Features with very small scores are regarded as weakly expressed in the learned representation and are therefore treated as candidates for exclusion before constructing the alphabet and training the forecasting models in Wayeb. The following architecture is presented on the following figure Figure 3. 2.

- **Input layer**
  Input dimension: input_dim= $F \times w$ (flattened sliding window)
  Fully connected layer: Linear(input_dim, 256)
  BatchNorm1d(256)
  ReLU activation
- **Encoder – hidden layer 1**
  Linear(256, 128)
  BatchNorm1d(128)
  ReLU activation
- **Encoder – hidden layer 2**
  Linear(128, 64)
  Linear(64, 32)
  BatchNorm1d(32)
  ReLU activation
- **Encoder – bottleneck layer**
  Linear(32, latent_dim)
  Output: latent representation $z \in \mathbb{R}^{latent\_dim}$
- **Decoder – hidden layer 1**
  Linear(latent_dim, 32)
  BatchNorm1d(32)
  ReLU activation
- **Decoder – hidden layer 2**
  Linear(32, 64)
  Linear(64, 128)
  BatchNorm1d(128)
  ReLU activation
- **Decoder – hidden layer 3**
  Linear(128, 256)
  BatchNorm1d(256)
  ReLU activation
- **Output layer**
  Linear(256, input_dim)
  Output dimension: same as input, reconstruction of the flattened sliding window.

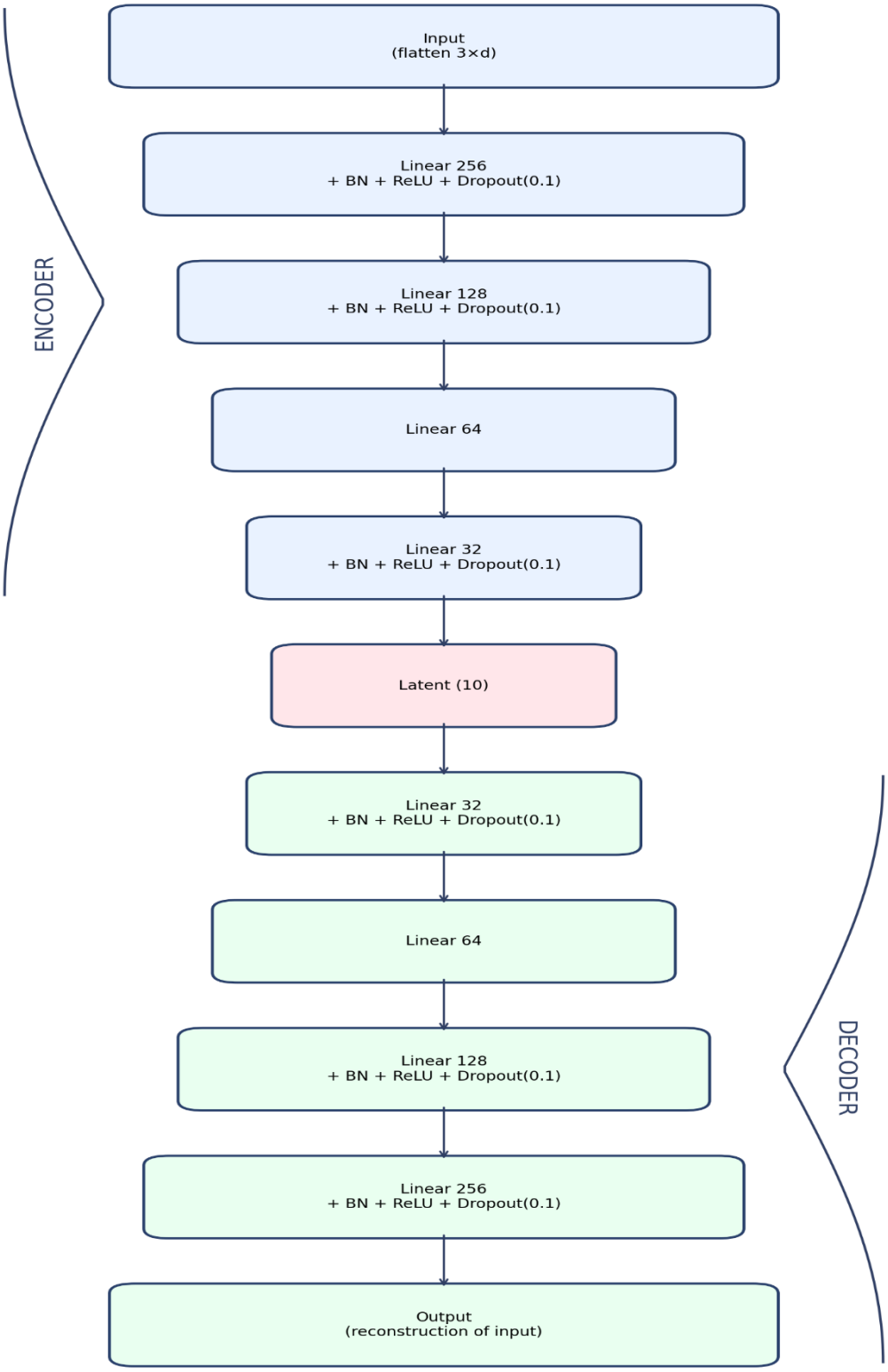

Figure 3. 2 Autoencoder Detailed Architecture

The chosen architecture allows the autoencoder to capture rich, non-linear structure in the data while enforcing a meaningful compression of the input. Starting from the high-dimensional flattened vector of size $F \times w$, the successive fully connected layers in the encoder gradually reduce the dimensionality and mix information across both features and temporal positions within the window. The use of ReLU activations introduces non-linearity and sparsity, enabling the network to model complex dependencies that cannot be captured by linear methods, while batch normalization at each hidden layer stabilizes training and reduces sensitivity to the scale and distribution of individual inputs. The bottleneck layer with dimensionality latent_dim acts as a compact latent representation in which the model is forced to retain only the most salient patterns necessary for reconstructing the original sliding window. The decoder, which mirrors this structure in reverse, learns to reconstruct the input from this compressed code. Successful reconstruction implies that the latent representation has preserved the essential information present in the original sequence of events. In this way, the architecture not only denoises and compresses the data, but also integrates information across features and time steps, making it suitable for deriving feature importance scores that reflect how much each input dimension contributes to the overall structure learned by the model.

For the pattern training of the autoencoder the following parameters are used:

- Number of epochs: 100,
- Learning rate: 0.0001,
- Mean Squered Error loss for back propagation,
- Adam Optimizer.

The following sliding window is applied in each pattern:

Pattern	Sliding Windows (events)
Merchant Mcc Diff	3
Decline Before Success	3
Card Limit	3

As a baseline for evaluating whether the features selected by the methods described above are indeed meaningful, we exploit the fact that our data are generated synthetically. The dataset generator allows us to explicitly control how each pattern is implemented and to specify which attributes follow particular distributions for each pattern. This provides a ground truth regarding the truly informative features and their behaviour, against which we can compare the outcomes of the feature selection methods.

4.5 Aggregation of Feature Selection Methods

To obtain a global, unified notion of feature importance and thus select the most appropriate features in a consistent way, we implemented a simple aggregation scheme that combines the outputs of all four methods, mutual information, χ^2, Pearson's correlation coefficient and the autoencoder-based scores. Each of these techniques measures relevance using a different criterion and scale, so relying on a single method risks over-emphasising one particular

aspect while ignoring others. By aggregating them into a single score per feature, we aim to exploit their complementary strengths and reduce the dependence of the final ranking on any one specific metric.

Concretely, for each feature we first collect its raw importance scores from the four methods and align them into vectors over the common set of features. Since the absolute magnitudes of the scores are not directly comparable across methods (for example, χ^2 values are several orders of magnitude larger than MI or PCC), we apply a rank-based normalization separately to each method: features are sorted by their score, the best feature receives a normalized value close to 1, and the worst a value close to 0, with the rest linearly spaced in between. For Pearson's coefficient we work with absolute values so that positively and negatively correlated features are treated symmetrically. The four normalized scores are then combined into a single aggregated importance value via a weighted average (with equal weights in our experiments), yielding one final score per feature as shown in the following formula:

$$S_j = \frac{w_{MI}\tilde{s}_j^{(MI)} + w_{x^2}s_j^{(x^2)} + w_{PCC}s_j^{(PCC)} + w_{AE}s_j^{(AE)}}{w_{MI} + w_{x^2} + w_{PCC} + w_{AE}}$$

Where:

- $j \in \{1, \dots, n\}$ index the features
- $m \in \{MI, x^2, PCC, AE\}$ index the methods
- $\tilde{s}_j^{(m)}$ is the raw importance score of feature j from method m
- $\tilde{s}_j^{(MI)} \in [0, 1]$ is the rank – normalized score for feature j from method m
- $w_m \geq 0$ is the weight assigned to method m here all weights are equal to one.

Sorting the features by this aggregated score produces a unified ranking that reflects the consensus of all four selection criteria and serves as the basis for choosing the subset of features to be used in the forecasting pipeline.

4.6 Complex Event Forecasting Setup

As discussed above, the framework that implements complex event forecasting (CEF) in our work is Wayeb. In this section, however, we do not revisit the underlying theory or the internal processing pipeline of Wayeb, as these have already been presented. Instead, we focus on how the experimental results are obtained and which configurable parameters of the framework we adjusted in order to produce the final plots.

Firstly, to obtain results that are both easily interpretable and useful, we aimed to make the experimental procedure as robust as possible. To this end, for each pattern considered, we ran multiple experiments while varying the main configurable parameters of the forecasting model, namely the confidence threshold, spread, order, and declaration score. After applying the feature selection methods, which will later be compared against the ground truth, we proceed to perform the forecasting runs. For each method, we evaluate all possible combinations of the configurable parameters within the following

ranges:

- Declaration: starting from zero extra predicates to most relevant in each pattern,
- Order: depends on the pattern {2, 3, 4} or {2, 4, 6}
- Spread: 5-6 depends on the pattern and 15
- Confidence Threshold = {0.1, 0.3, 0.5, 0.7, 0.9}

For the evaluation of the wayeb the following metrics are going to be plotted with respect to the parameters above:

- Accuracy
- Recall
- Precision
- F1 – Score
- Negative predictive value (precision of other class)
- Specificity (recall of other class)
- Informedness
- Matthews Correlation Coefficient
- Confusion matrices

All of the above metrics will help us analyze the different dynamics of the forecasting model and demonstrate how an appropriate feature selection strategy can introduce a new level of flexibility and effectiveness into the overall pipeline.

To summarise, this section has described the complete feature-selection stage and the experimental configuration of the complex event forecasting framework. We presented three classical filter-based methods (PCC, mutual information and χ^2), a dense autoencoder used as an embedded neural feature selector, and an aggregation scheme that combines their outputs into a unified ranking of features. We also detailed how temporal information is reintroduced through sliding windows, how the autoencoder is trained, and how Wayeb is configured in terms of its main parameters and evaluation metrics. In the following section, we turn to the empirical results, where we compare the selected features against the synthetic ground truth and analyse how different feature-selection strategies affect forecasting performance across the patterns under study.

5 Results

In this section we present the experimental evaluation of the proposed feature-selection–based alphabet reduction approach within the Wayeb framework. We systematically examine how varying the selected features and corresponding declaration sets affect automaton complexity, runtime performance, and forecasting quality across the synthetic fraud datasets.

5.1 Evaluation Methodology

This subsection details the experimental design and evaluation protocol used to assess the proposed feature-selection–driven alphabet reduction within the Wayeb framework. The objectives are twofold:

1. quantify how selecting a subset of attributes and the ensuing declaration budget influence symbolic alphabet size, SDFA complexity, and computational cost and
2. measure the downstream effect on forecasting quality.

Experiments span multiple synthetic fraud scenarios with distinct underlying temporal patterns. For each scenario, we vary the number of selected features via top-k lists (where relevant, score thresholds), and we sweep key Wayeb hyperparameters (order, spread, confidence, and forecasting horizon). The compared approaches include a full-specification baseline without feature selection and a family of FS-based configurations at increasing declaration budgets as already discussed. This design allows us to expose trade-offs between structural complexity and predictive performance, and to identify “knee” points where additional declarations yield diminishing returns.

The evaluation follows a fixed procedure. For each dataset and configuration, we compute feature rankings using the selected criteria, derive the declaration set, generate predicates and minterms, construct and determinize the SDFA, and run recognition and forecasting on held-out streams. We collect structural metrics (alphabet size, number of states and transitions) together with task metrics. Results are aggregated across repeated runs with fixed random seeds to control variability and reported as means with dispersion where appropriate.

5.2 Feature Selection Results

The current subsection reports the feature-selection results that drive the declaration budgets used later in this chapter. For each dataset, we present bar plots of normalized importance scores per feature for the individual criteria considered (e.g., correlation-based scores, mutual information, χ^2, and the autoencoder-derived signal), with features ordered from most to least important. To make the scores comparable across methods, each vector is scaled to [0,1] and ties are broken by the raw (pre-scaled) value. Alongside the per-method profiles, we compute an aggregated ranking by combining the normalized scores into a

single importance measure and highlight the resulting top-k sets that are carried forward to automaton construction. We comment on consistencies and discrepancies across methods, and where applicable, relate high-ranking attributes to the pattern-defining variables in the synthetic scenarios. The goal is to verify that the selected subsets are both compact and semantically plausible before assessing their effect on alphabet size, SDFA complexity, runtime, and forecasting quality.

The following figures represent the importance scores of different methods discussed before. For each method we plot at the x axis the feature names and in y axis the importance score. We start with the pattern named “Merchant Mcc Diff” and each corresponding dataset.

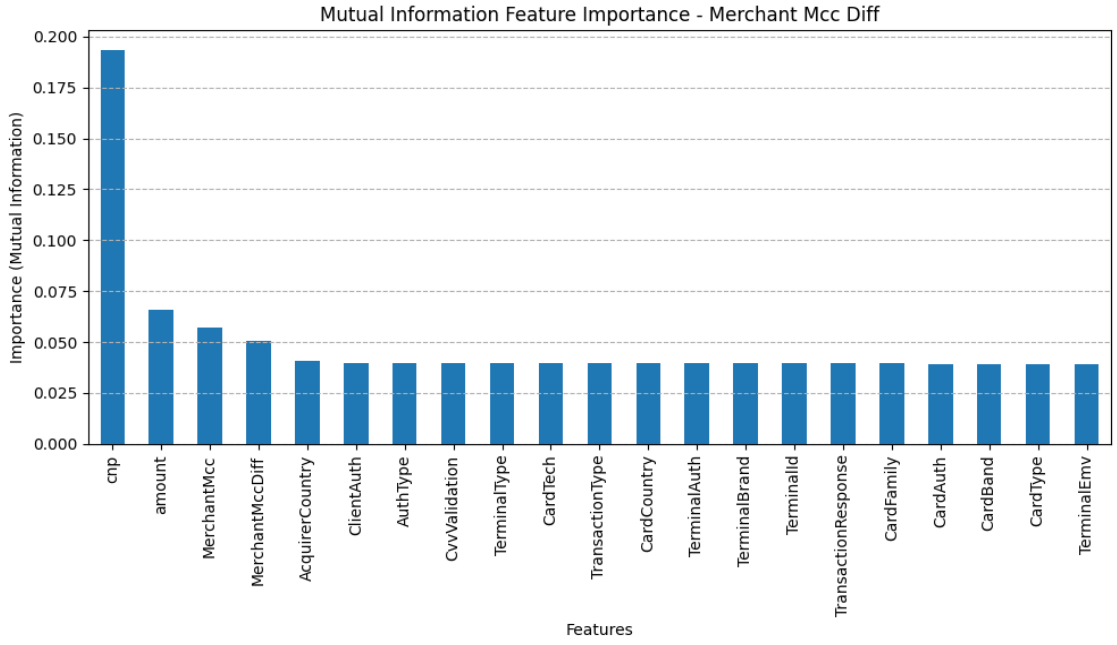

Figure 5. 1 Mutual Information Feature Importance Score for Pattern Merchant Mcc Diff

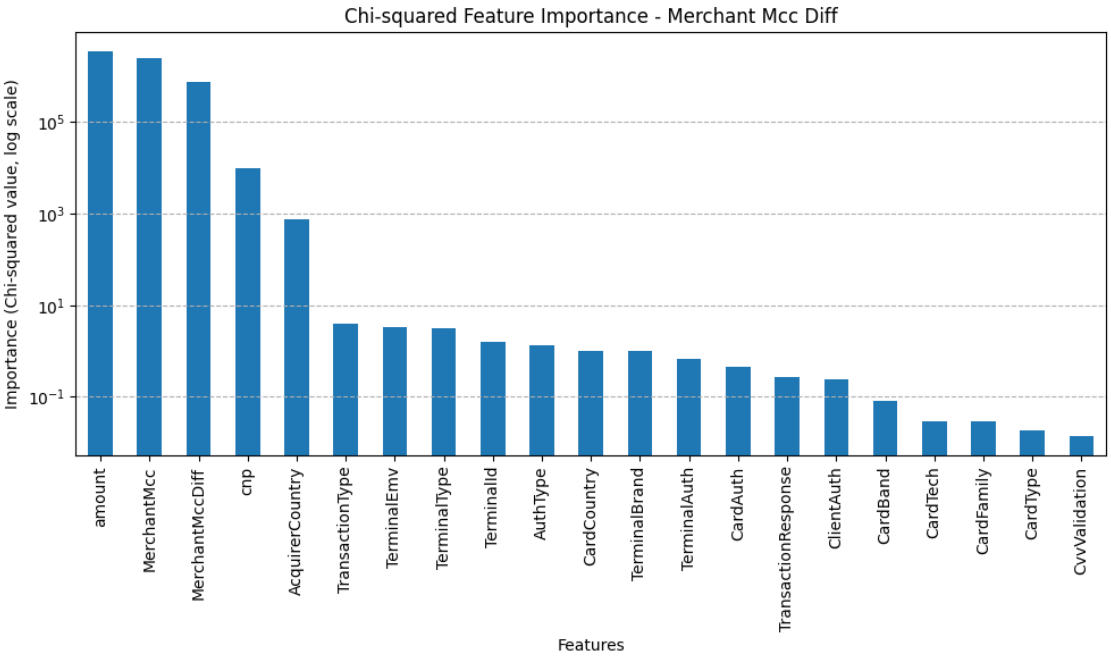

Figure 5. 2 Chi - Squared Feature Importance Score for Pattern Merchant Mcc Diff

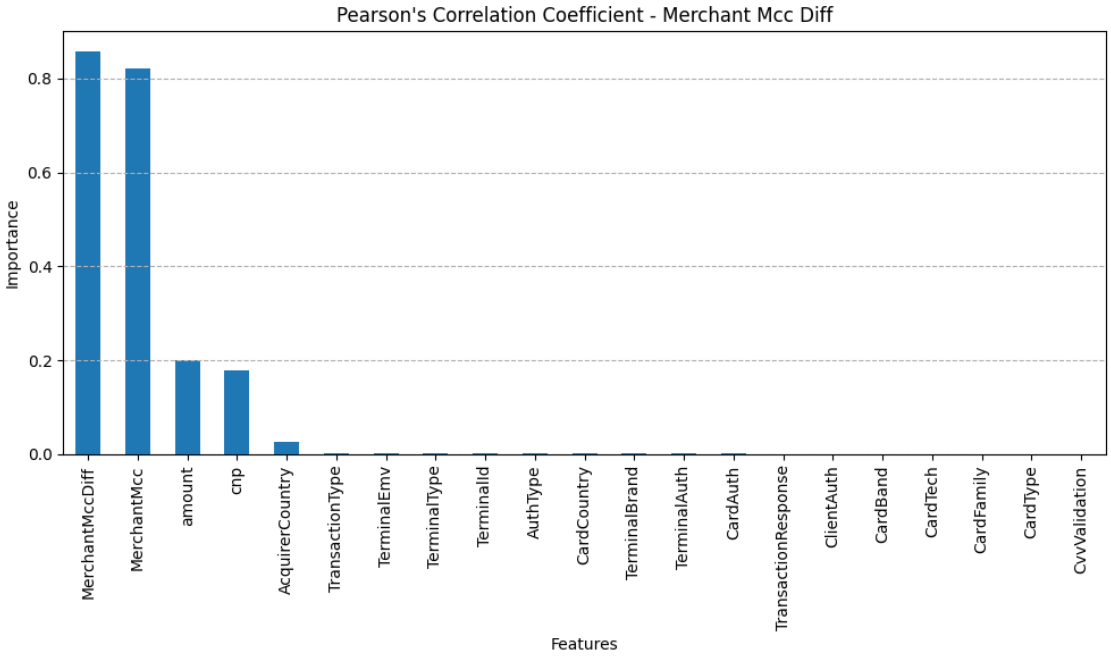

Figure 5. 3 Pearson's Correlation Coefficient Feature Importance Score for Pattern Merchant Mcc Diff

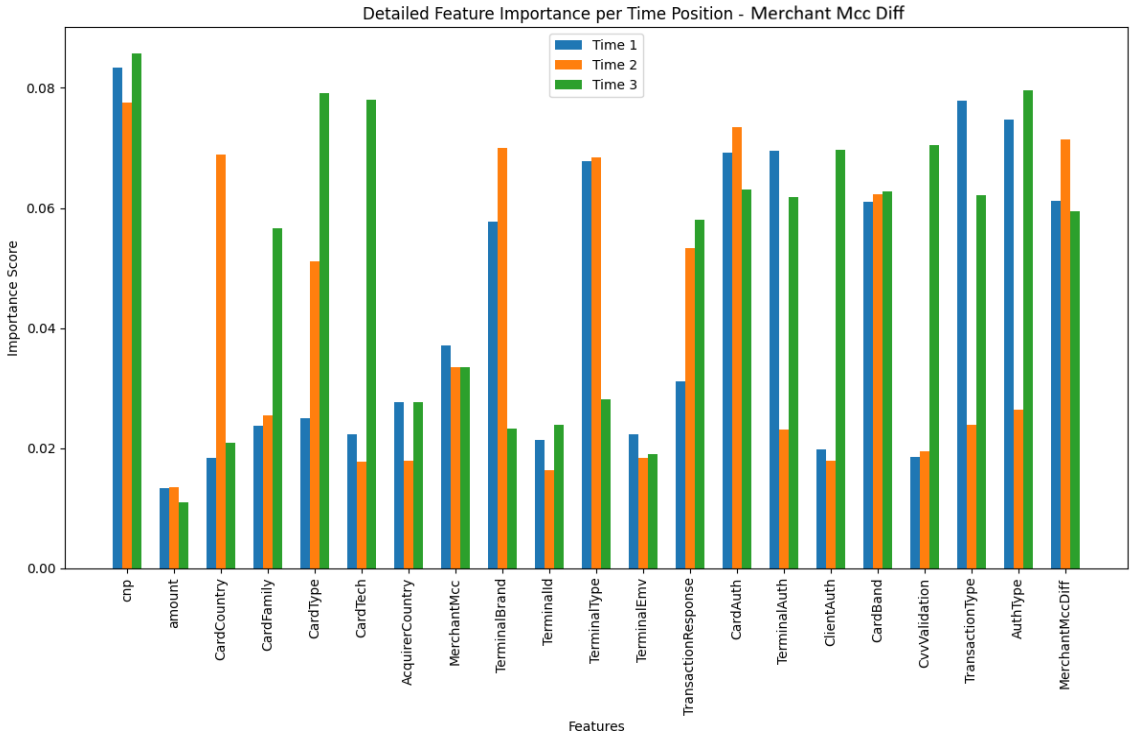

Figure 5. 4 Autoencoder Weights per Time Slot for Pattern Merchant Mcc Diff

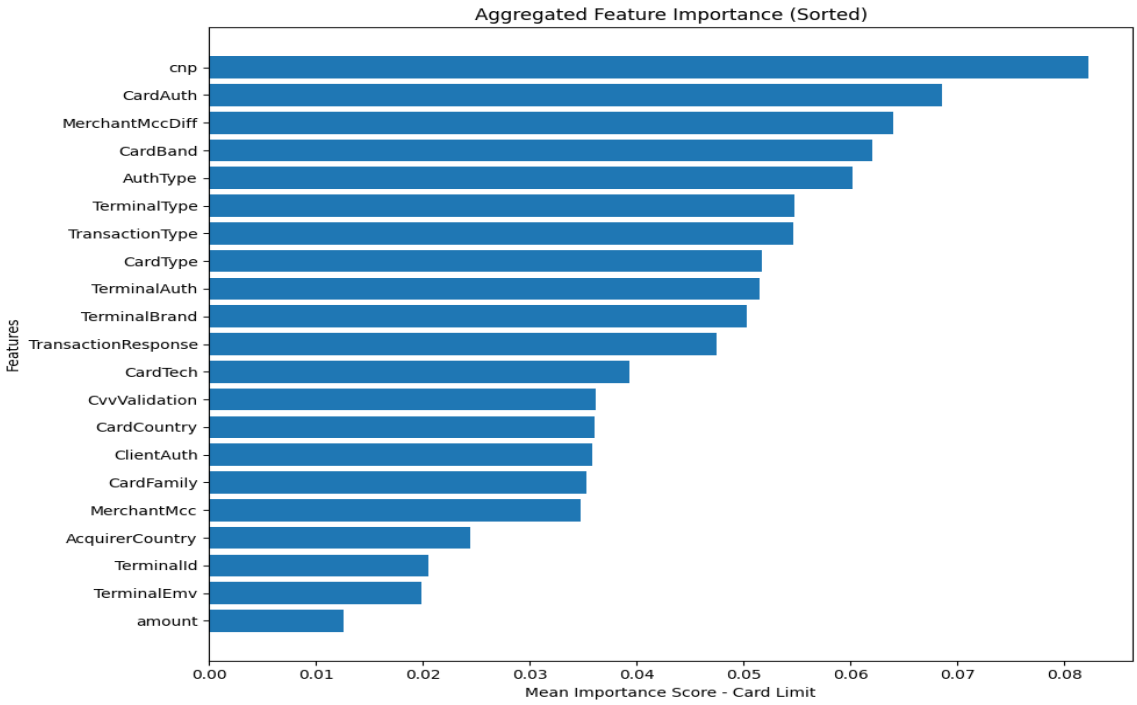

Figure 5. 5 Autoencoder Aggregated Weights for Pattern Merchant Mcc Diff

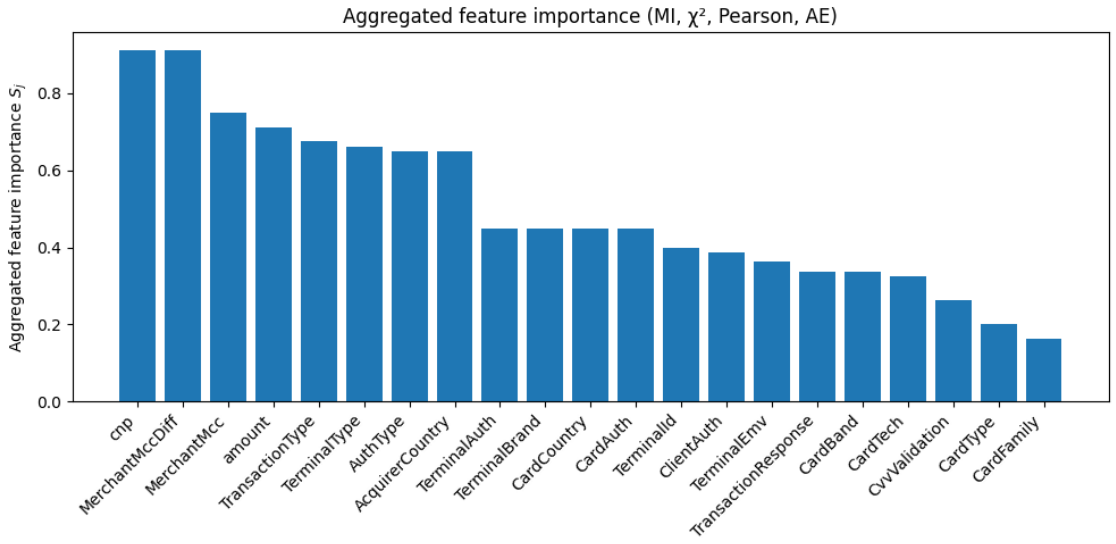

Figure 5. 6 Aggregated Feature Importance Score from all Methods for Pattern Merchant Mcc Diff

Since the datasets are generator-based and the pattern is hand-crafted, we know which attributes actually drive the fraud logic. For Merchant Mcc Diff, the decisive variables are MerchantMccDiff (explicitly in the SRE), MerchantMcc (declared later in the SRE), amount (explicitly in the SRE) and cnp (declared later in the SRE). Any selector that captures signal faithfully should rank these near the top, other attributes may appear due to correlation or proxy effects.

Mutual information(MI) as we see in Figure 5. 1 places cnp first, followed by amount, MerchantMcc, and MerchantMccDiff, with most remaining variables forming a low, almost flat tail. This is expected because MI rewards any non-linear dependence and is insensitive to monotonic rescaling, so a binary flag such as cnp can score highly when it co-occurs with the fraudulent sequences produced by the generator. Amount also scores well because the fraudulent windows concentrate on specific ranges, making their distribution distinguishable from the background. Other metadata appear low because their distributions are similar across classes or only weakly correlated with the crafted pattern.

The χ^2 scores shown in Figure 5. 2 are on a log scale and amount tops the list by a wide margin, with MerchantMcc and MerchantMccDiff next, then a sharp drop to cnp and the rest. Chi 2 is driven by differences in frequency distributions and is sensitive to the number of distinct non-negative values: a heavy-tailed, strictly non-negative variable like amount will often produce very large χ^2 when the positive class occupies a distinct region of the range. MCC variables also show large χ^2 because their category frequencies shift strongly in fraudulent windows. The remaining attributes contribute little once these primary shifts are explained.

Next is Pearson correlation at Figure 5. 3. Here MerchantMccDiff and MerchantMcc dominate ($\approx$0.86 and $\approx$0.82), which aligns tightly with the ground truth. Amount and cnp follow at a distance, while most other features are near zero. This behavior reflects Pearson's focus on linear association with a binary label and the fact that the class-conditional codes of MCC variables shift markedly under the pattern. Because these categorical fields are encoded as integers, large class-specific shifts look strongly "linear" to Pearson, balanced or multi-modal categorical fields naturally collapse toward zero.

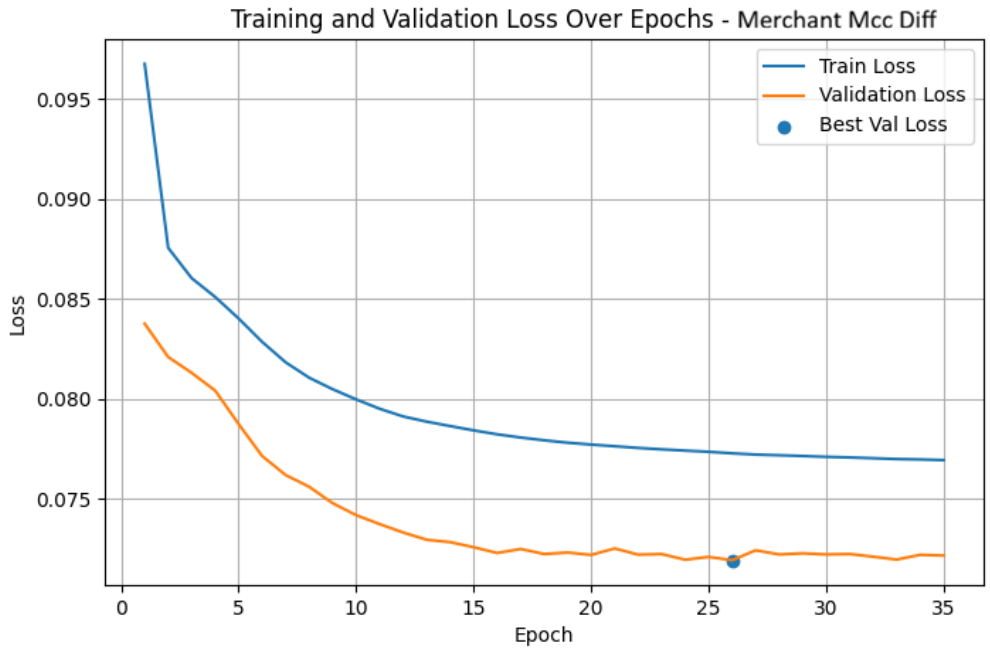

Figure 5. 7 Autoencoder Training - Validation MSE Loss Over Epoch for Pattern Merchant Mcc Diff

The autoencoder, as we explained at section 4.4.1 takes a sliding window of three consecutive events (window_size=3, stride=1), flattens them into a single vector, and learns to reconstruct that vector with an MSE loss. All inputs are MinMax-scaled to [0,1] (no special handling for amount), so every feature contributes on the same numeric scale to the reconstruction error. The network allocates capacity to whatever dimensions most reduce loss, feature "importance" in the Figure 5. 5 is computed directly from the trained model as the mean absolute value of the first encoder layer's weights for each input position, which is a standard saliency proxy for how much the model needs each input to reconstruct the window.

Against that setup, Figure 5. 4 shows a clear story. Binary "switch" and change variables, most notably cnp and TransactionResponse, related signals—carry the highest weights across the three time slots, with a consistent rise toward the most recent slot (Time 3). Under MinMax scaling and MSE, 0↔1 flips produce sharp local errors if the bottleneck fails to pass them through, so gradients concentrate on these coordinates, BatchNorm and mild Dropout further encourage the encoder to reserve dedicated channels for such decisive toggles rather than spread capacity thinly. By contrast, slowly varying or quasi-static descriptors (e.g., card and terminal metadata) are easier to reconstruct from context and thus receive lower weights, there is less penalty if the latent ignores small movements on those axes. The time profile itself reflects the underlying dynamics of the episodes: the tail of the fraudulent run tends to occur in the latest positions of each 3-step window, so the encoder relies more on Time-3 (and often Time-2) copies of response/authentication features than on their older counterparts. In short, given MinMax scaling, an MSE objective, and a narrow latent (10), the model prioritizes discrete transitions and recent evidence. That is why cnp and

response-pathway variables dominate the bars, while smooth or almost constant fields need little dedicated capacity to keep reconstruction error low.

Figure 5. 7, which represents the training of autoencoder, shows a fast initial decrease and then a smooth plateau, with the best validation point epoch 26. Validation remains slightly below training throughout, which is consistent with the presence of Dropout and BatchNorm in the network Noise injected during training raises the training loss, while at evaluation time Dropout is disabled and BatchNorm uses its stabilized statistics, yielding a lower validation loss. The absence of a late divergence between the two curves indicates that the autoencoder is not overfitting and has settled into a stable reconstruction regime. This gives us confidence that the signal we extract for feature selection reflects systematic structure in the windows rather than memorization artifacts.

The final ranking averaging MI, χ^2, Pearson, and AE with equal weights and places cnp narrowly first, with MerchantMccDiff and MerchantMcc immediately behind, and amount as the next strongest numeric driver. A second tier includes Transactional data, Auth Type, and Acquirer Country, followed by terminal/auth metadata and card properties. TransactionResponse sits lower, as expected for this pattern. The high position of cnp, is consistent with its distinct class distribution and the AE's sensitivity to late-window changes, so we treat it as a contextual proxy rather than a predicate. Overall, the outcome aligns with expectations MCC-related variables (and amount) emerge as the core features, while the remaining attributes provide auxiliary signal.

Having completed the analysis for Merchant Mcc Diff, we now turn to "Decline Before Success" pattern. The following figures show the per-method feature-importance profiles for this pattern and highlight the top-k subsets used next.

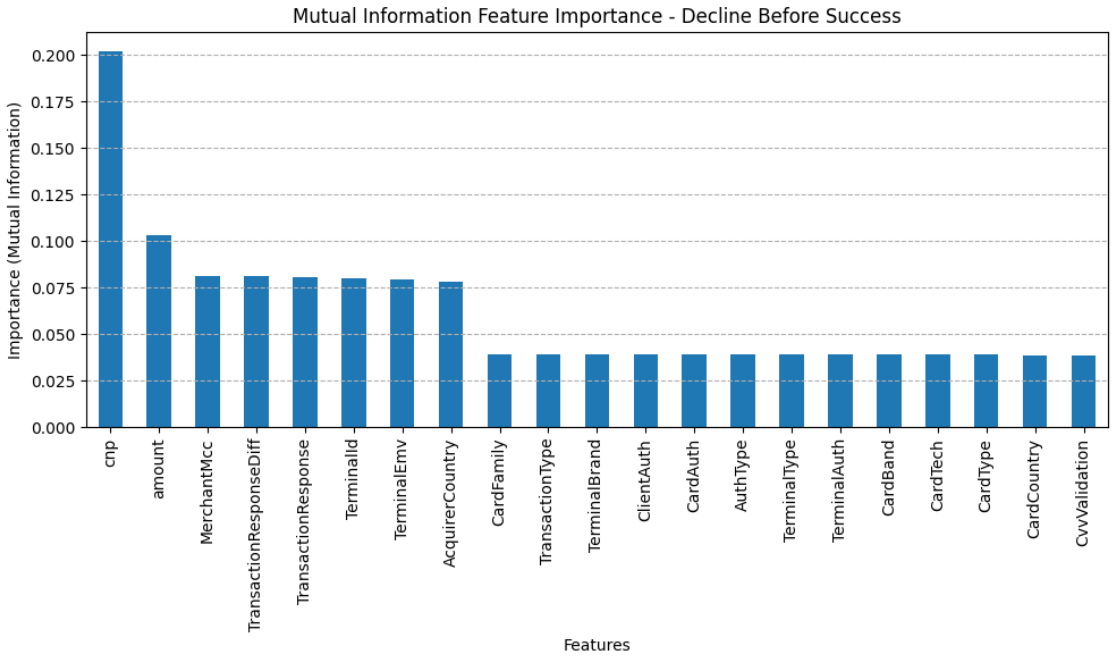

Figure 5. 8 Mutual Information Feature Importance Score for Pattern Decline Before Success

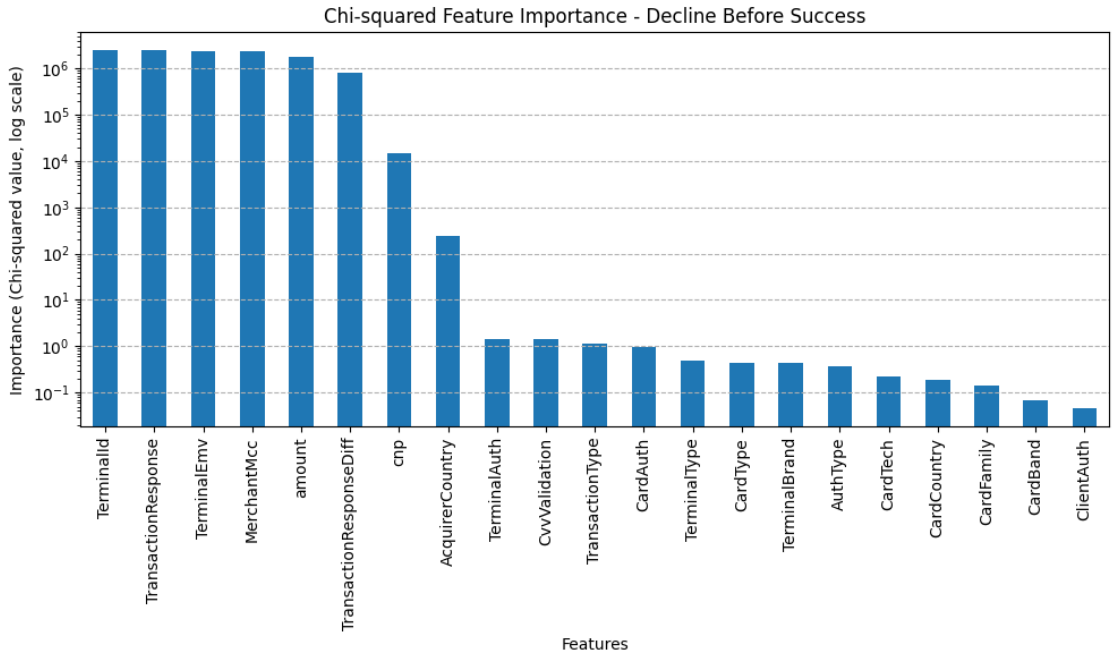

Figure 5. 9 Chi - Squared Feature Importance Score for Pattern Decline Before Success

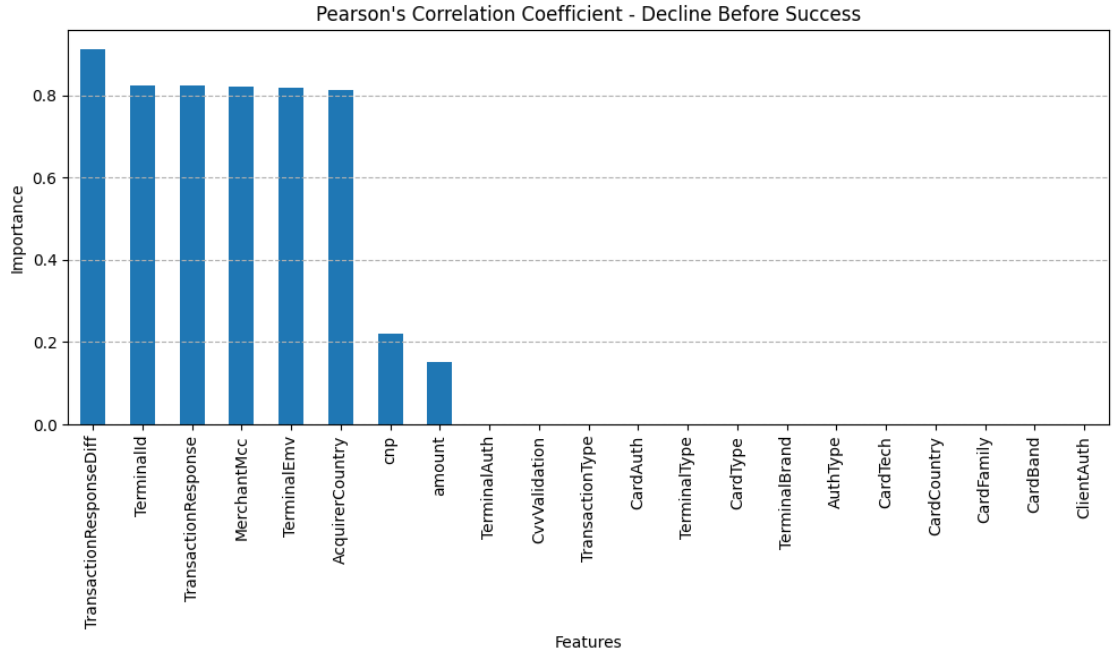

Figure 5. 10 Pearson's Correlation Coefficient Feature Importance Score for Pattern Decline Before Success

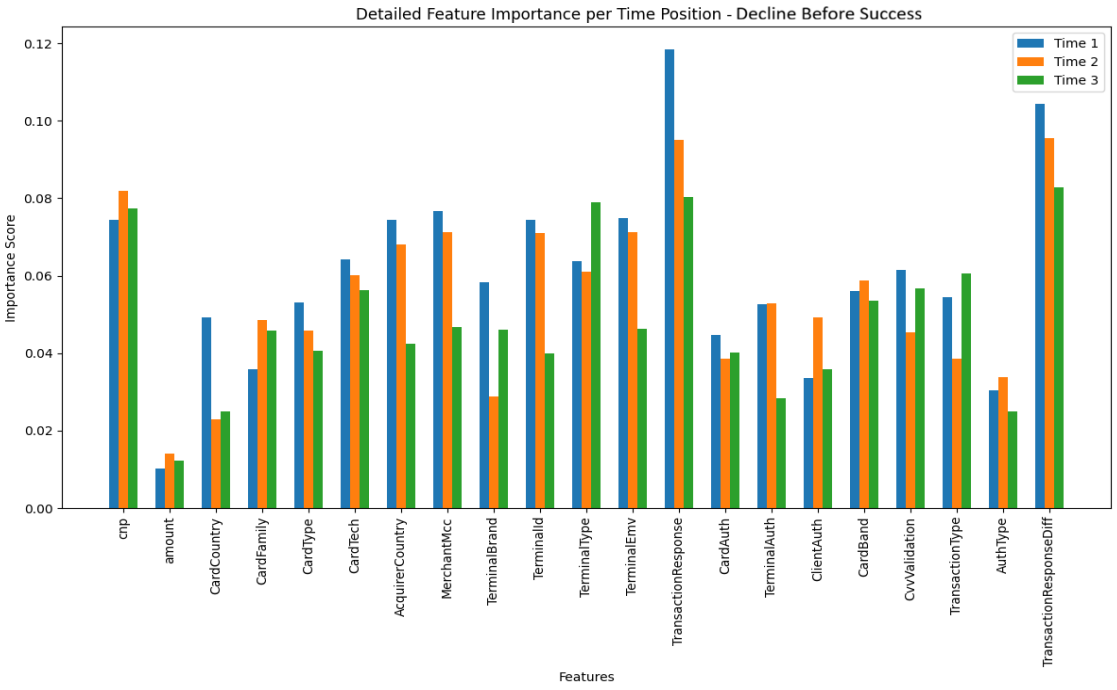

Figure 5. 11 Autoencoder Weights per Time Slot for Pattern Decline Before Success

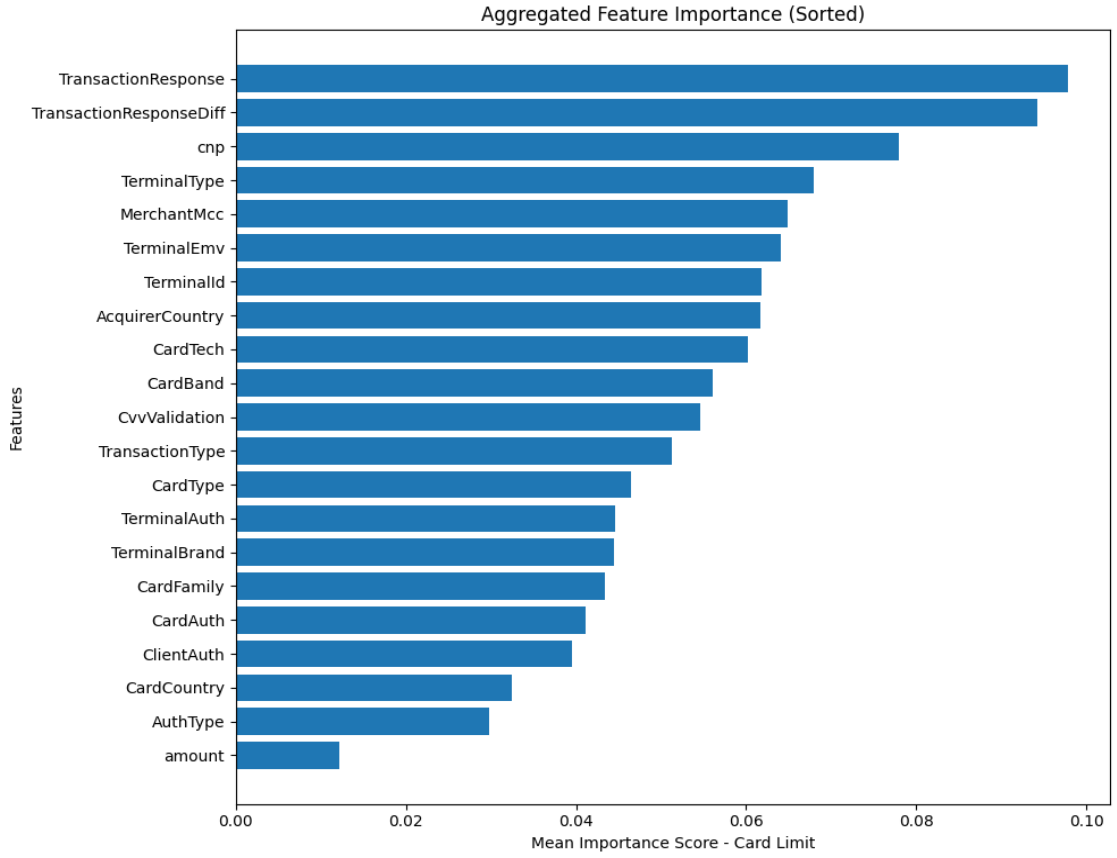

Figure 5. 12 Autoencoder Aggregated Weights for Pattern Decline Before Success

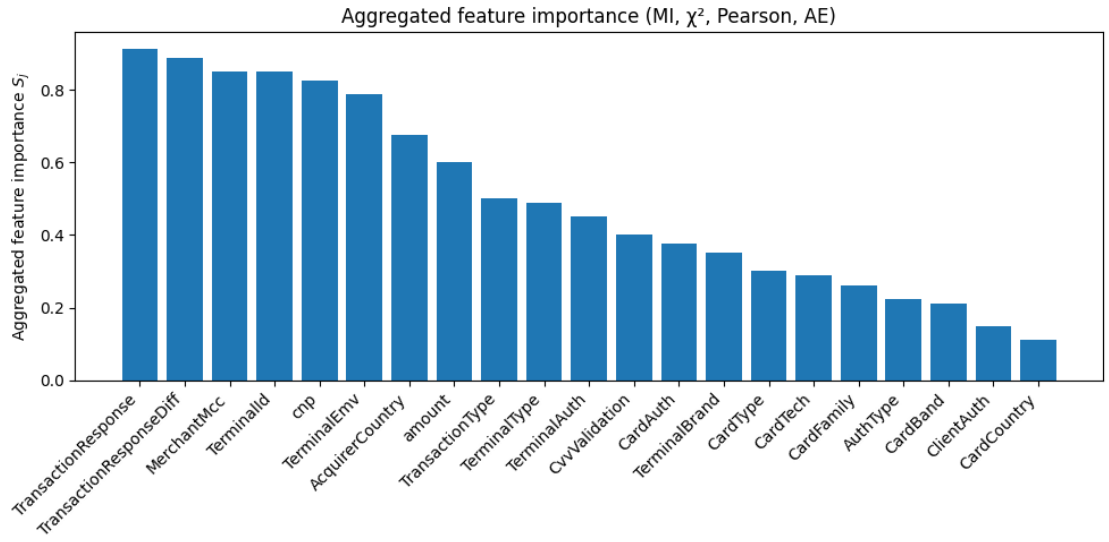

Figure 5. 13 Aggregated Feature Importance Score from all Methods for Pattern Merchant Decline Before Success

Mutual Information at Figure 5. 8 rises when a feature's distribution looks noticeably different between positive and negative windows. That's why the binary cnp flag can sit very high. Positives are strictly card-present while the background mix includes both types, so the pattern of {0,1} values changes a lot across classes. amount also shows up because the "declines then a larger final amount" shape produces a distinctive range in positive windows. Outcome fields (TransactionResponse, Diff) also appear near the top because the label is tied to how the response evolves.

Chi2 at Figure 5. 9 explodes for TransactionResponse / Diff (the decline→success switch makes the class tables very uneven) and for high-cardinality identifiers like TerminalId, TerminalEmv, MerchantMcc because positive windows reuse the same codes across the six events, concentrating counts in a few categories. That makes them look extremely "different" across classes—useful for screening, but not necessarily causal.

Pearson's Correlation Coefficient at Figure 5. 10 prefers features whose average numeric value differs cleanly between classes. It naturally favors the outcome fields and often shows high values for integer-coded IDs (TerminalId, MerchantMcc, TerminalEmv) because treating codes as numbers creates a difference in class averages. By contrast, cnp and amount can rank lower here if their relationship to the label isn't close to a straight-line trend.

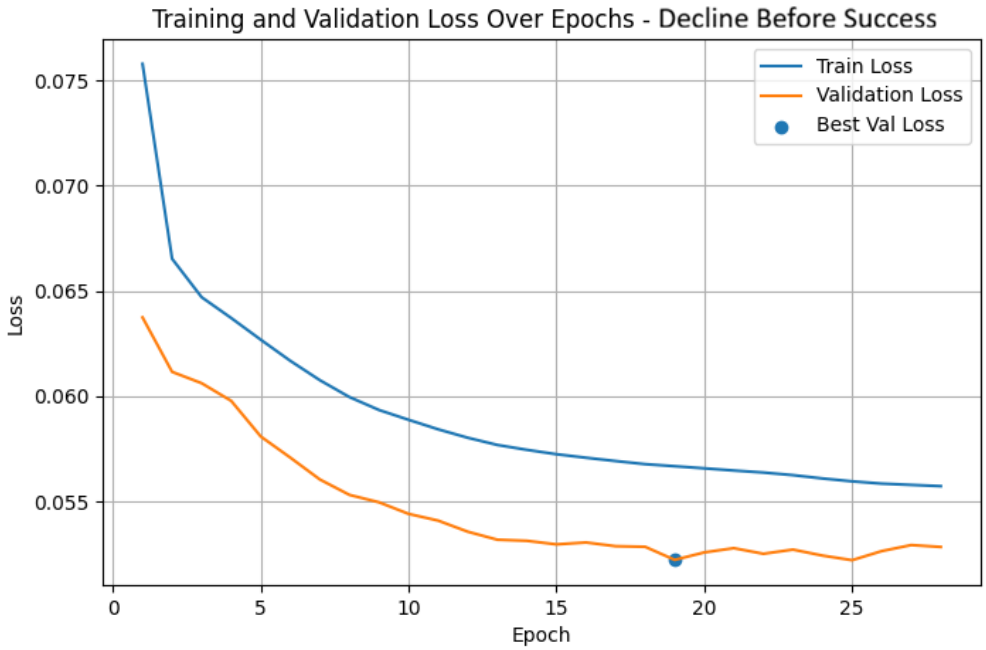

Figure 5. 14 Autoencoder Training - Validation MSE Loss Over Epoch for Pattern Decline Before Success

In Figure 5. 11 the weights per time – slot are discrete and percise. Variables that encode discrete regime changes, TransactionResponse and its finite-difference variant TransactionResponseDiff, dominate the ranking, followed by cnp and terminal-pathway fields (e.g., TerminalEmv/TerminalType). With MinMax scaling and an MSE loss, 0↔1 flips create sharp localized errors if the bottleneck fails to transmit them, gradients therefore concentrate on these coordinates, and BatchNorm + mild Dropout encourage the encoder to dedicate capacity to such decisive toggles. The time breakdown also matches the episode dynamics: TransactionResponse and TransactionResponseDiff peak at the earliest slot in the 3-step window, then taper (Time 1 > Time 2 > Time 3). This asymmetry is consistent with windows that often straddle the "decline" phase just before the success, making the earliest symbol most difficult to predict, hence the higher weight. This logic is applied also later to the forecaster and is based on the following simple rule:

- "The start of a fraudulent pattern cannot be modeled"

Based on that rule, the first time slot is the most difficult by the model to be predicted so the weights are higher. Conversely, slowly varying or quasi-static descriptors (card/terminal metadata, issuer geography, etc.) carry lower weights. They are predictable from context and impose little penalty if encoded coarsely. In short, given MinMax scaling, an MSE objective, and a narrow latent (10), the model prioritizes discrete transitions and recent context that characterize the DBS tail—hence the tall bars for TransactionResponse/TransactionResponseDiff and the comparatively modest contribution of static profile attributes.

The training/validation loss curve for the autoencoder presented at Figure 5. 14 shows a rapid early drop followed by a gentle plateau, with the best validation point at epoch 19. Throughout training, the validation loss stays slightly below the training loss—expected with Dropout and BatchNorm: stochastic masking

during training elevates the training loss, while at evaluation Dropout is off and BatchNorm uses stabilized running statistics, lowering the validation loss. The lack of a late separation between the curves suggests no overfitting and a stable reconstruction regime, indicating that the feature-importance signal reflects genuine structure in the windows rather than memorization.

The final ranking Figure 5. 13 shows the aggregation of all methods with equal weights. Transaction Response and Diff are the attributes which dominates the importance score, which aligns with the ground truth. The next tier includes terminal variables and merchant mcc which is true in pattern while cnp still be in top 5 features even its not part of the pattern. As we described above the nature of a binary attribute and its values distribution make cnp score high. Overall the feature selection was accurate and gone as expected with an exception on cnp.

Finally, we turn to the last pattern, **Card Limit**. The figures that follow present the per-method feature-importance profiles for this pattern (features on the x-axis, normalized scores on the y-axis) and highlight the top-k subsets we carry forward to the declaration budgets for automaton construction and evaluation.

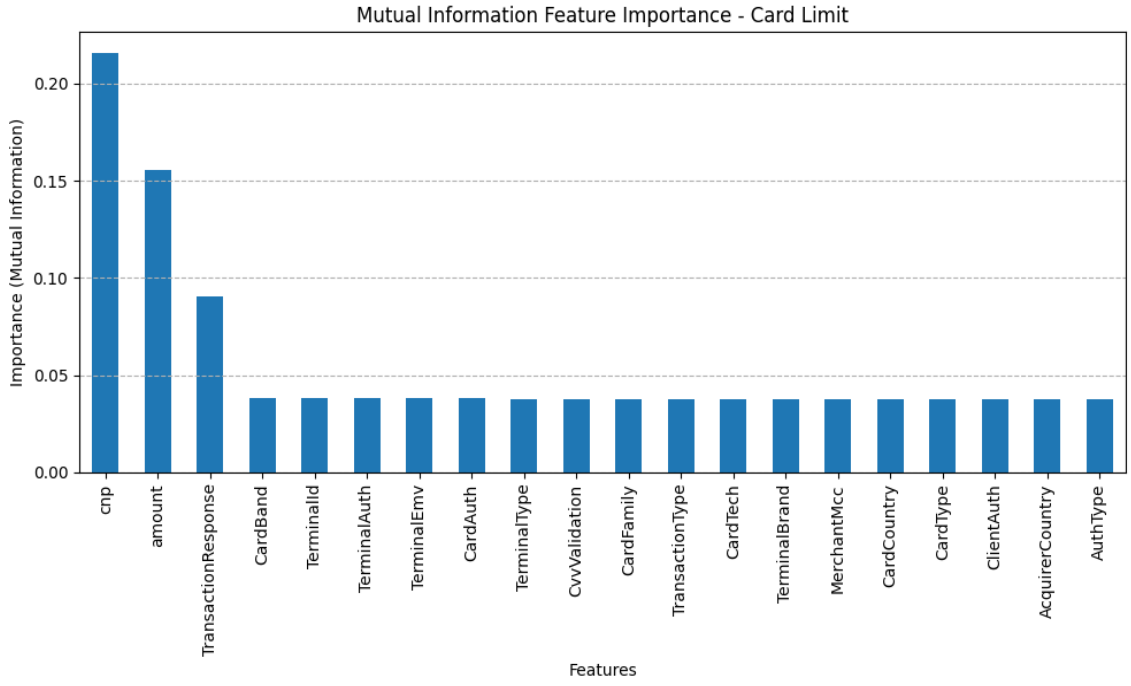

Figure 5. 15 Mutual Information Feature Importance Score for Pattern Card Limit

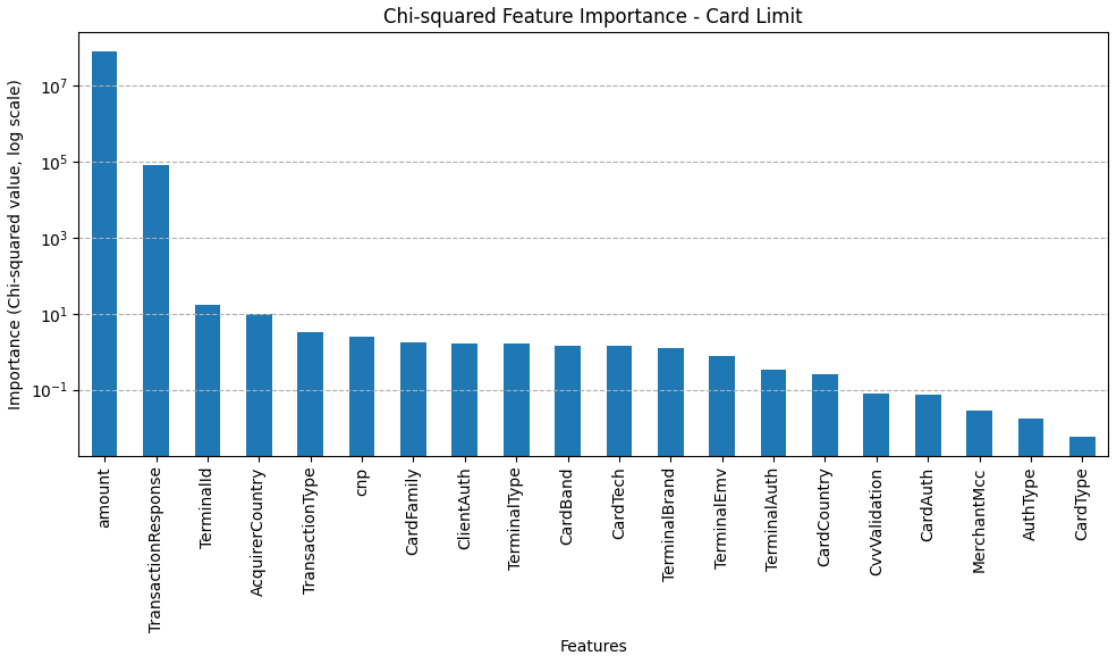

Figure 5. 16 Chi - Squared Feature Importance Score for Pattern Card Limit

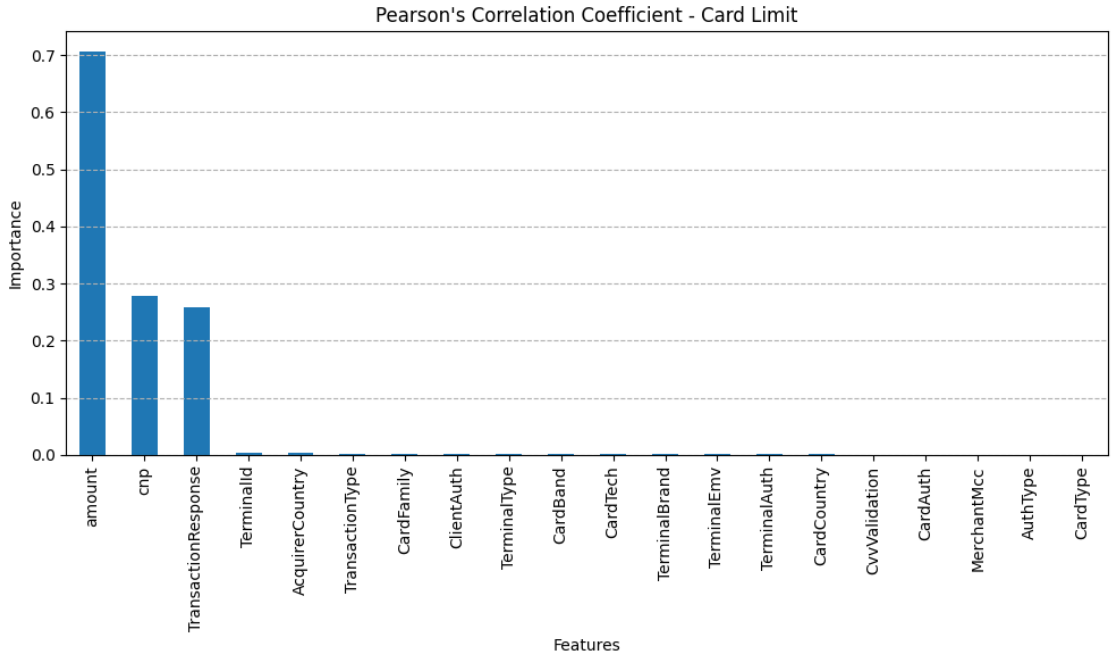

Figure 5. 17 Pearsons Correlation Coefficient Feature Importance Score for Pattern Card Limit

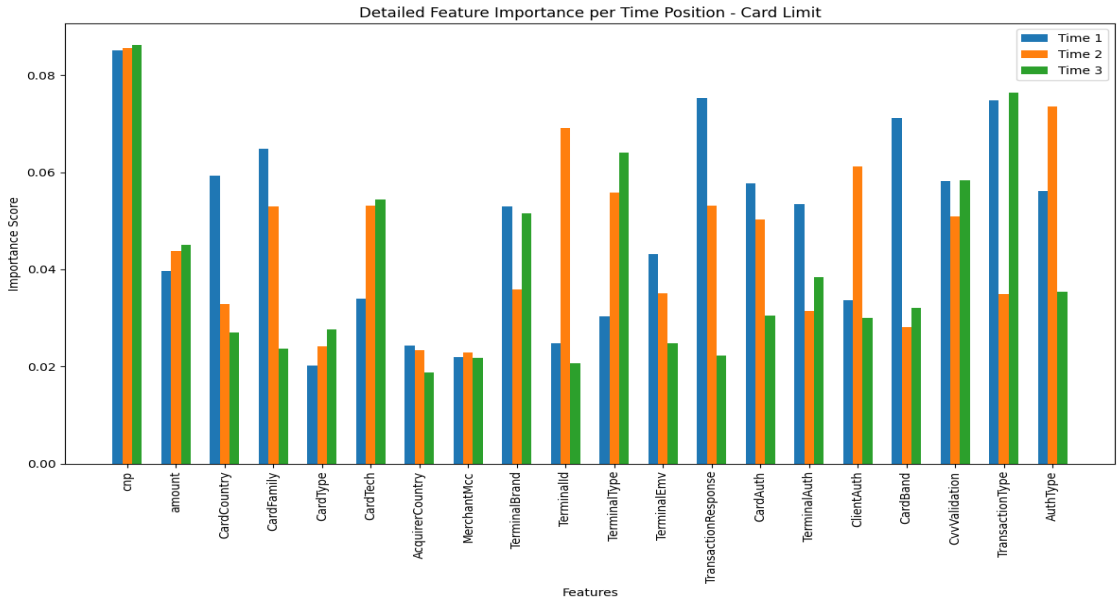

Figure 5. 18 Autoencoder Weights per Time Slot for Pattern Card Limit

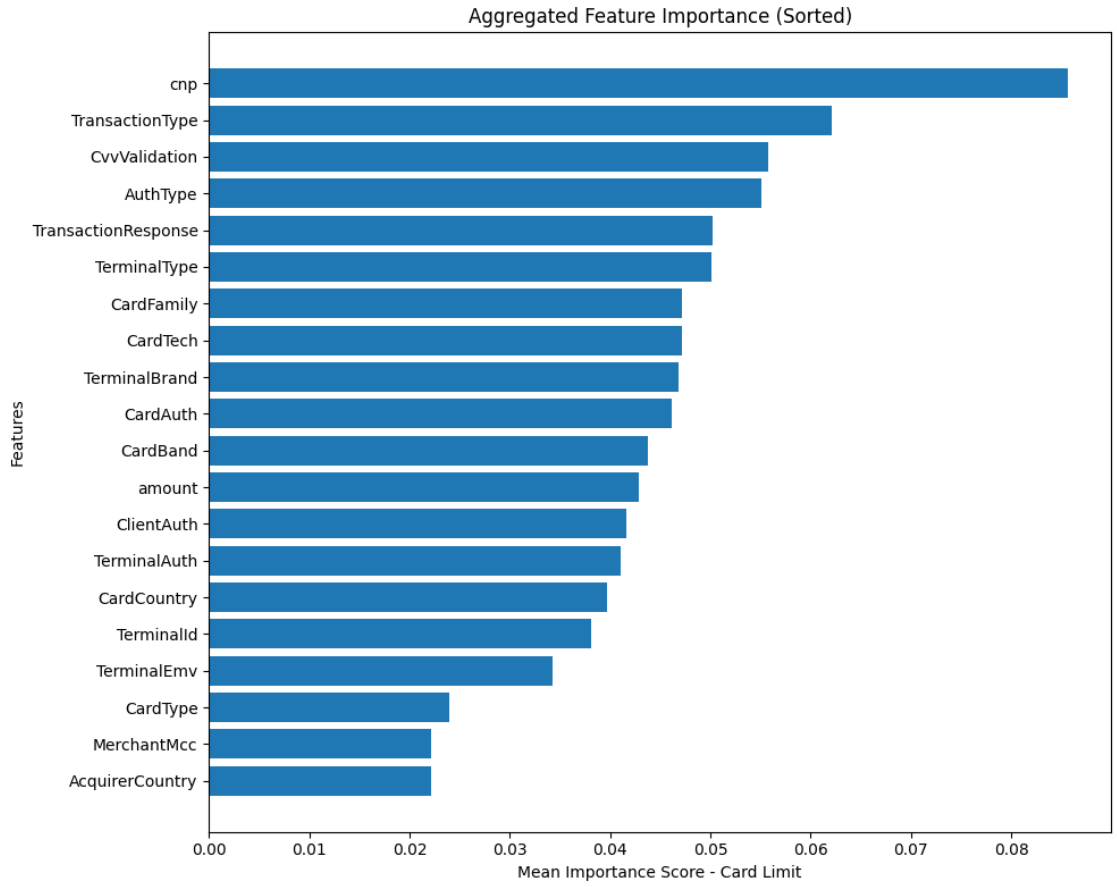

Figure 5. 19 Autoencoder Aggregated Weights for Pattern Card Limit

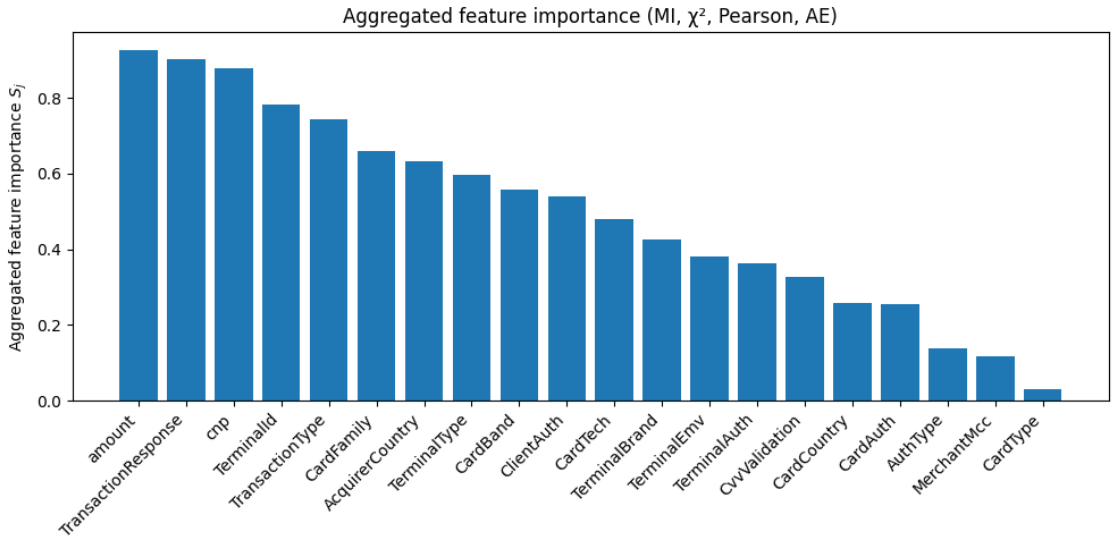

Figure 5. 20 Aggregated Feature Importance Score from all Methods for Pattern Card Limit

At Figure 5. 15 MI ranks again cnp at the top which actually agrees highly with the observation done above. Amount stands out because limit events concentrate in a distinctive range, and TransactionResponse rises because approvals/declines are distributed differently around limit cases which perfectly aligns with the pattern. Most other variables form a flat tail, indicating modest distributional change.

Chi2 reacts to uneven category frequencies after discretization as shown at Figure 5. 16. It shows an orders-of-magnitude spike for amount (distinct bin usage near the limit), and very large values for TransactionResponse. High-cardinality identifiers such as TerminalId also climb because positive windows tend to reuse the same codes, concentrating counts in a few categories. This makes these features extremely effective for screening, though not necessarily causal.

At Figure 5. 17 Pearson favors features whose numeric averages separate the classes. amount dominates by a wide margin, with cnp and TransactionResponse clearly above the rest, nearly all other variables are near zero. This says the class means for amount differ strongly, while integer-coded IDs contribute little linearly once averaged.

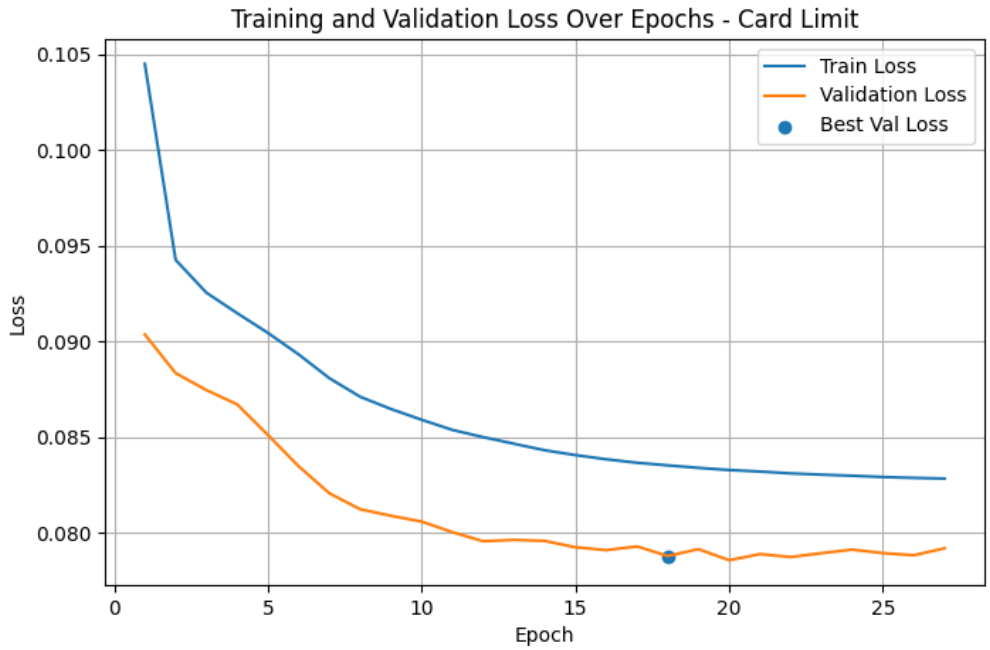

Figure 5. 21 Autoencoder Training - Validation MSE Loss Over Epoch for Pattern Card Limit

At Figure 5. 18 is the same read-out for the **Card Limit** pattern, using the identical autoencoder configuration (window_size = 3, stride = 1), MinMax scaling to [0, 1] for all inputs, MSE reconstruction loss, and a 10 - dimensional bottleneck. Feature "importance" is the mean absolute weight of each input coordinate in the first encoder layer (a standard saliency proxy), coordinates that the network must pass through to keep reconstruction error low receive larger weights.

Furthermore, Figure 5. 18 in combination with Figure 5. 19 show a different temporal fingerprint than in Decline Before Success. The strongest, most stable signal is cnp, which ranks at or near the top in all three time slots—typical for a binary switch whose 0↔1 flips are expensive to reconstruct under MSE once everything is scaled to [0, 1]. Several response/authentication pathways also carry substantial weight but peak at earlier positions in the window. TransactionResponse is highest at Time 1 and then decays, CardAuth and TerminalAuth are also front-loaded, whereas ClientAuth peaks at Time 2. This "early-evidence" pattern is consistent with card-limit episodes, where authorization decisions and response codes begin to diverge from the baseline before the limit breach fully manifests. In contrast, TerminalType (and to a lesser extent CardBand and CvvValidation) retain sizable mass toward Time 3, suggesting that terminal/context cues help the model finalize the reconstruction near the tail of the episode. As before, slowly varying descriptors (card/terminal identities, country codes, merchant category) sit lower, they are either near-constant within a 3-step window or predictable from the surrounding context, so

the encoder does not need to dedicate much capacity to them.

The autoencoder's loss curves in Figure 5. 21 show a steep initial decline followed by a shallow plateau, with the best validation value reached around epoch 19. Across training, validation remains marginally lower than training—consistent with Dropout and BatchNorm: stochastic masking inflates the training loss, whereas at evaluation Dropout is disabled and BatchNorm relies on settled running means/variances, yielding a lower loss. The absence of any late divergence between the curves indicates no overfitting and a stable reconstruction regime, suggesting that the derived feature-importance signal captures genuine structure in the windows rather than memorized artifacts.

The final equal - weight aggregate places amount first and Transaction Response second, with cnp and TerminalId close behind, and TransactionType rounding out the top tier. This pattern is expected: amount is favored by Pearson and χ^2, TransactionResponse is favored by all three classical filters, and cnp earns high MI (distributional shift) and AE (late reconstruction difficulty). In practice, amount and TransactionResponse are the core features to prioritize in the declaration budget, cnp, terminal IDs/EMV, and transaction/auth types are auxiliary features whose inclusion can be decided via small ablations or by using less bias-prone encodings for IDs.

5.3 Automata Alphabet Reduction Results

Building on the feature-selection stage, we fix compact declaration sets per pattern. For Merchant Mcc Diff and Card Limit we retain the top-2 features, and for Decline Before Success we retain the top-5. Using these declarations, we compile the corresponding Wayeb models and sweep key hyperparameters—most notably the confidence threshold θ and the model order PST/PSA order where relevant, we also vary spread to characterise performance across operating points. The figures that follow report, for each pattern and sweep, both forecasting/recognition metrics (precision, recall, F1) and structural/runtime metrics (alphabet size). Each panel keeps the declaration set fixed for that pattern and shows how the metrics evolve as θ and order change, making the trade-offs and revealing "knee" regions where additional threshold/order increases yield diminishing returns. All runs use the same train/test protocol and environment to keep comparisons fair across patterns.

The following figures are visual representations of metrics extracted from wayeb related to the pattern "Merchant Mcc Diff" by the declaring as extra predicates "CNP" (decl=1) and "Merchant Mcc" (decl=2).

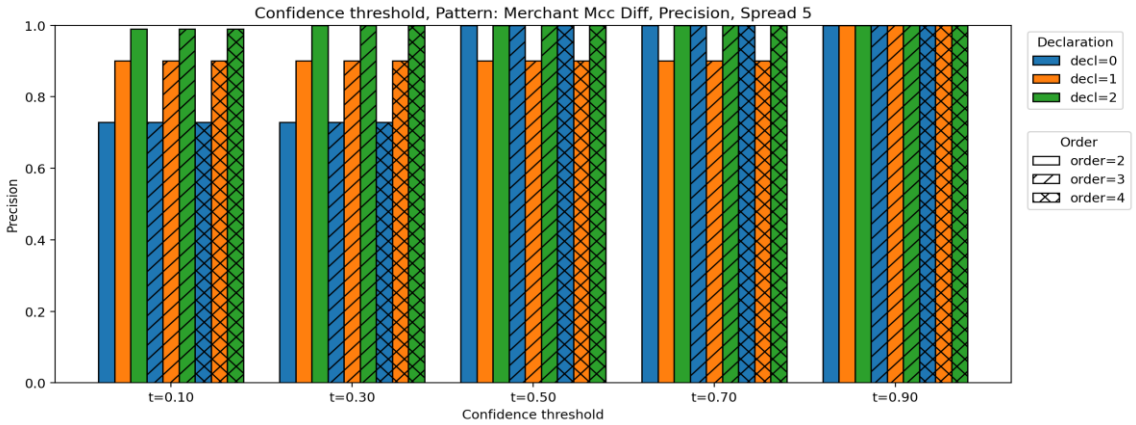

Figure 5. 22 Precision per Confidence Thresholds for Pattern Merchant Mcc Diff and Spread 5

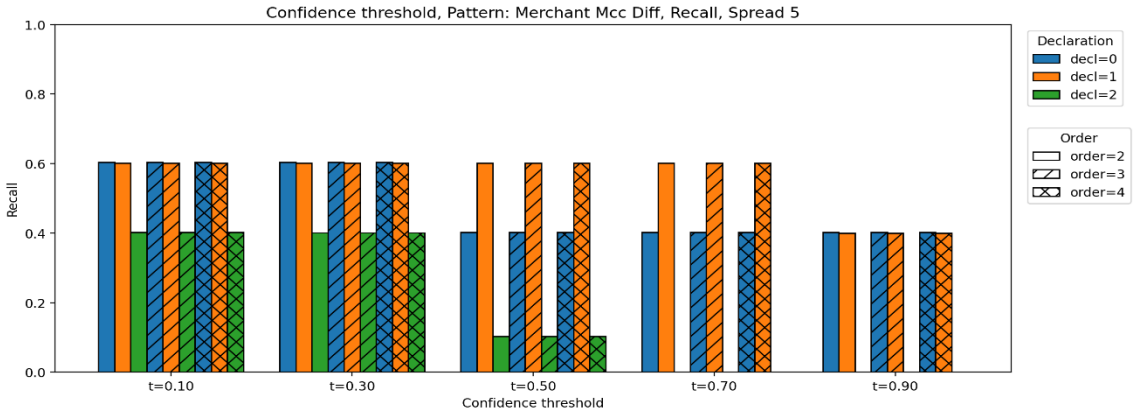

Figure 5. 23 Recall per Confidence Thresholds for Pattern Merchant Mcc Diff and Spread 5

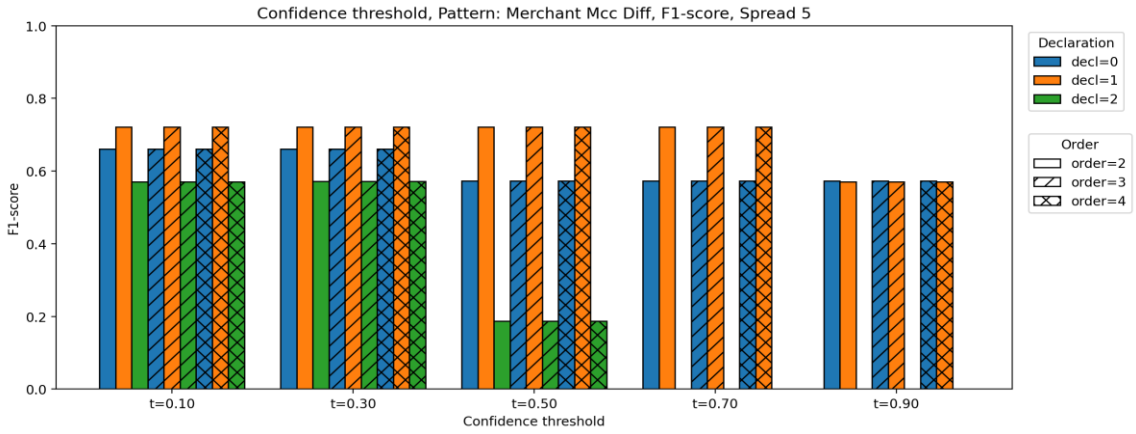

Figure 5. 24 F-1 Score per Confidence Thresholds for Pattern Merchant Mcc Diff and Spread 5

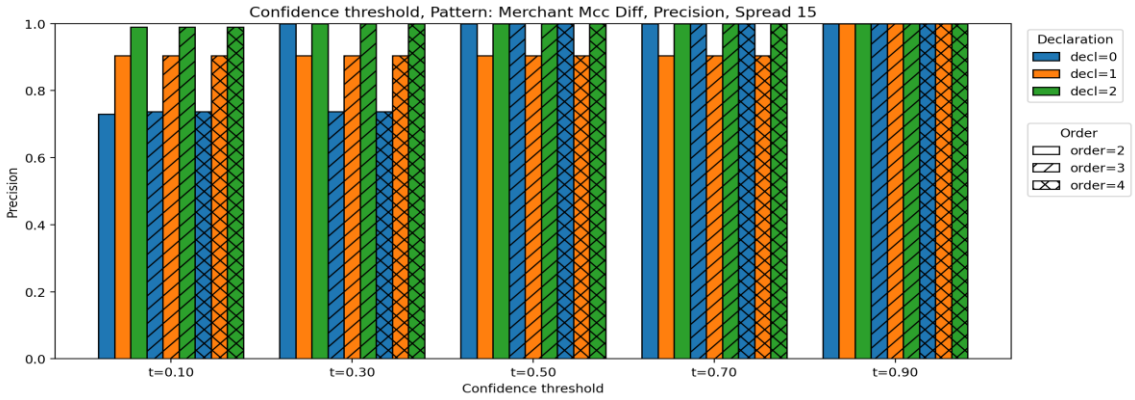

Figure 5. 25 Precision per Confidence Thresholds for Pattern Merchant Mcc Diff and Spread 15

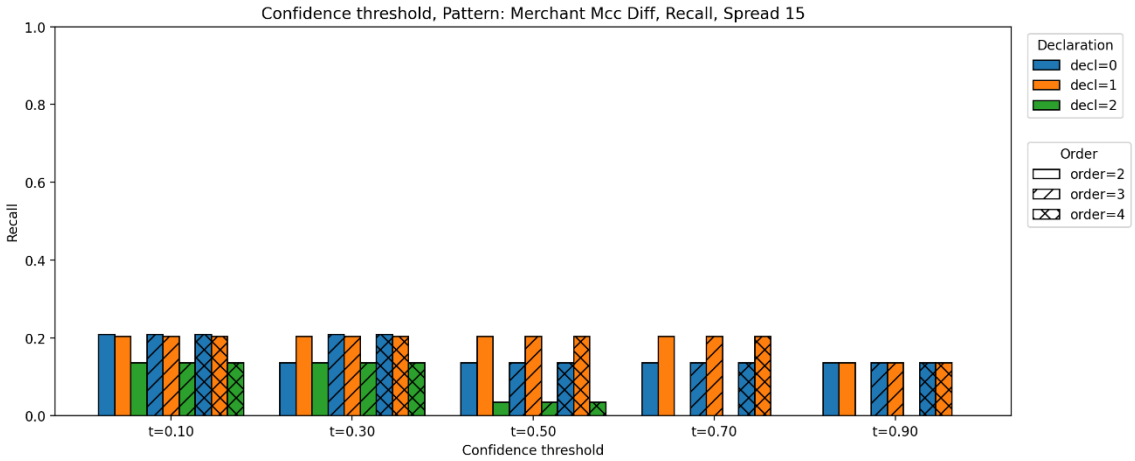

Figure 5. 26 Recall per Confidence Thresholds for Pattern Merchant Mcc Diff and Spread 15

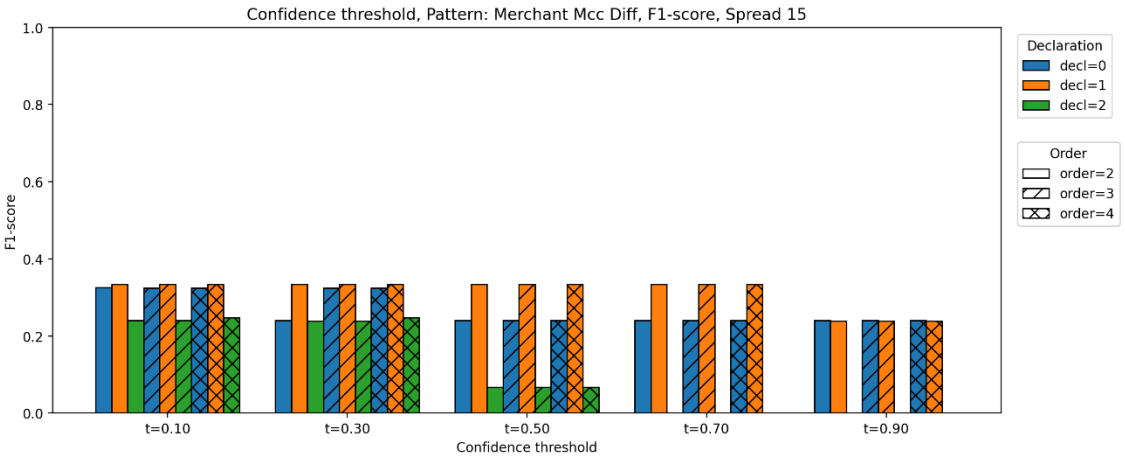

Figure 5. 27 F-1 Score per Confidence Thresholds for Pattern Merchant Mcc Diff and Spread 15

The figures (Figure 5. 22, Figure 5. 23, Figure 5. 24, Figure 5. 25, Figure 5. 26, Figure 5. 27) report precision, recall, and F1-score as functions of the confidence threshold for each forecasting spread, bar hatching encodes the model order (2, 3, 4). Two consistent trends appear. First, as the confidence threshold increases, recall drops and precision rises or stays high, so F1 typically peaks at low thresholds and declines thereafter. Second, enlarging the spread makes the task harder: the model must issue correct forecasts further ahead in the sequence.

The case spread = 15 is especially stringent. Because the pattern spans four consecutive events, evaluating with a 15-event horizon means that, per true occurrence, up to 15 forecasts may be attempted but only 4-time positions can possibly be counted as correct; the rest are scored as errors. This tight window explains why recall and F1 are generally low at spread 15, while precision remains high—the model fires rarely and, when it does, it is usually close to the true event (few false positives). Finally, higher orders (3–4) modestly stabilize the metrics across thresholds and spreads, indicating a small benefit from modeling longer temporal dependencies, but the spread and threshold choices dominate overall performance.

With spread = 5, the effect of alphabet reduction is pronounced. Introducing declarations improves the forecaster's behavior—both raising the probability mass near the true completion and increasing detection rates. The best recall is achieved by the baseline pattern (decl=0) and the first declaration (decl=1, TransactionResponse), which perform essentially identically: in the recall plot both reach ~0.60, meaning the model typically captures about 3 of the 5 forecastable positions within the five-event window of each fraudulent episode. This is close to the practical ceiling for this model, since a fraudulent run cannot be predicted reliably before it begins to manifest. The difference between decl=0 and decl=1 appears in precision: adding cnp to the alphabet reduces false positives, yielding a cleaner operating point without sacrificing recall. More broadly, no system can "pre-patternize" fraud to predict in advance that a specific PAN will be compromised; what can be done—and is achieved here—is to detect the tail of the episode early and with fewer spurious alarms.

Overall, the feature-selection pipeline improves the forecaster. Although it

does not advance the earliest detectable point of a fraud episode, it reduces false alarms and yields cleaner operating points. Declaring cnp and sweeping the confidence threshold from 0.1 to 0.9 shows a consistently confident model: detections begin to appear around t ≈ 0.7, indicating that posterior probabilities are generally high. Increasing the order further stabilizes these probabilities by injecting short-term memory, which is reflected in the figures—particularly for spread = 15, where higher order sustains better F1 and recall under stricter thresholds.

The following figures are visual representations of metrics extracted from wayeb related to the pattern “Decline Before Success” by the declaring as extra predicates

1. As individuals(decl=1): “Merchant Mcc”, “Termina Id”, “Terminal Emv”, “Transaction Response”, decl = mcc, decl = id, decl = emv and decl = tr respectively.
2. As combination of two: “Transaction Response” with “Terminal Id”, decl = tr_id.

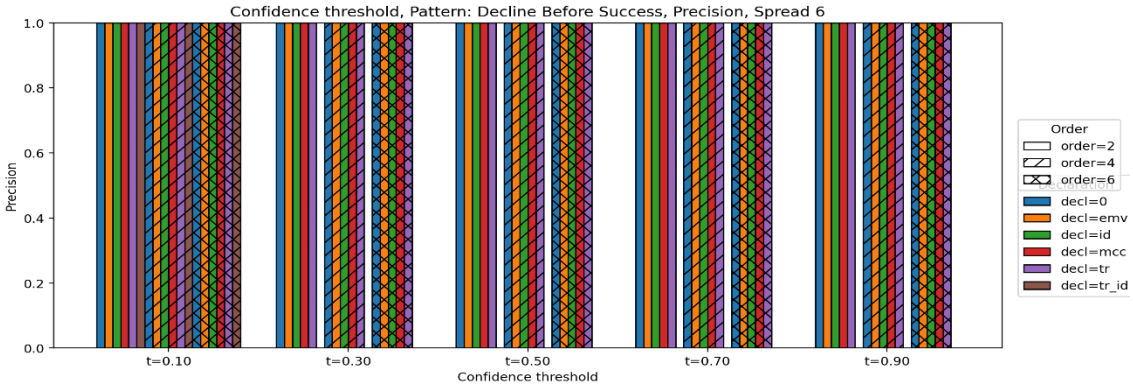

Figure 5. 28 Precision per Confidence Thresholds for Pattern Decline Before Success and Spread 6

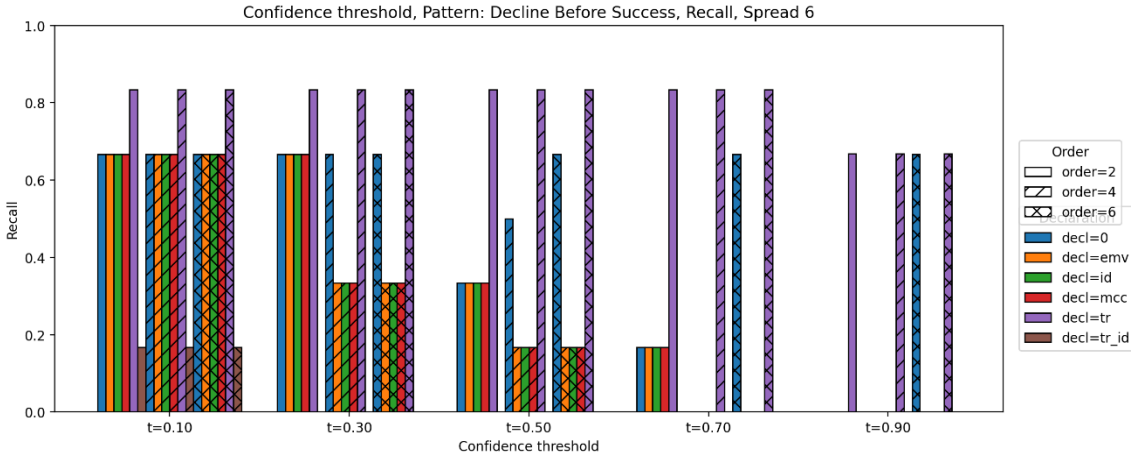

Figure 5. 29 Recall per Confidence Thresholds for Pattern Decline Before Success and Spread 6

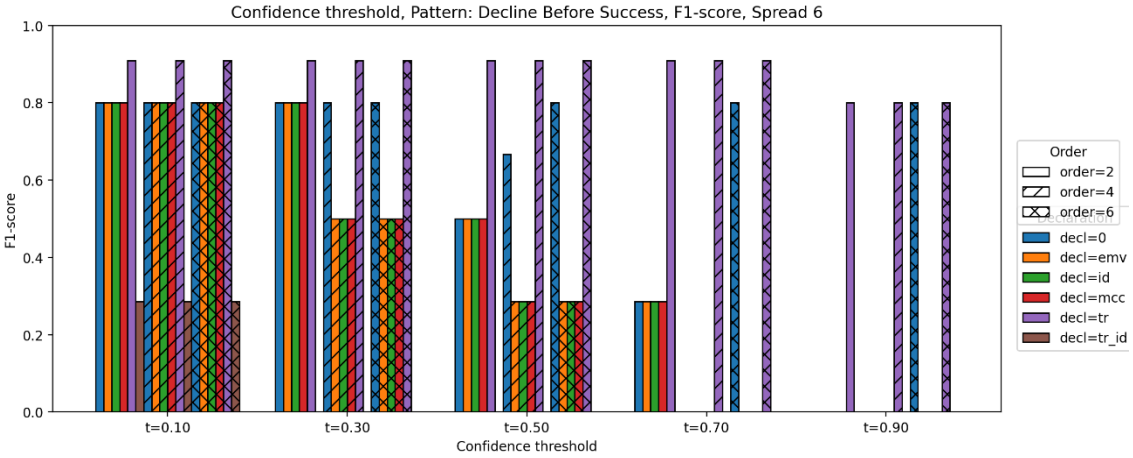

Figure 5. 30 F-1 Score per Confidence Thresholds for Pattern Decline Before Success and Spread 6

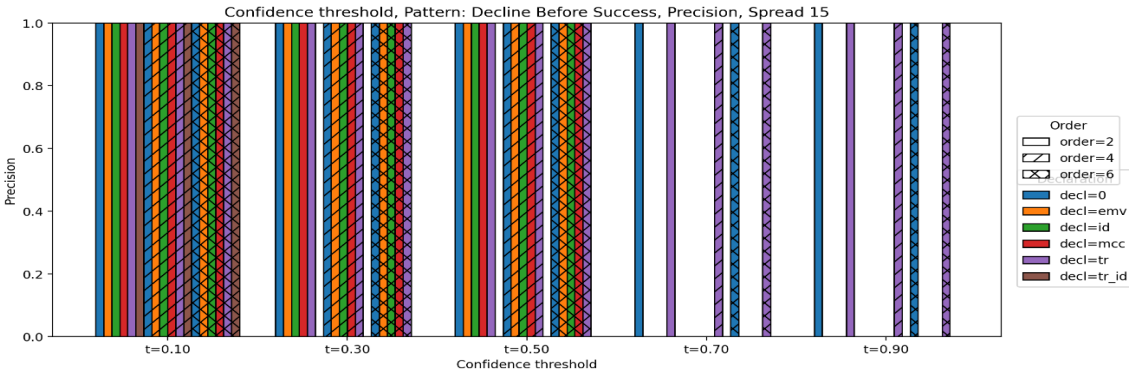

Figure 5. 31 Precision per Confidence Thresholds for Pattern Decline Before Success and Spread 15

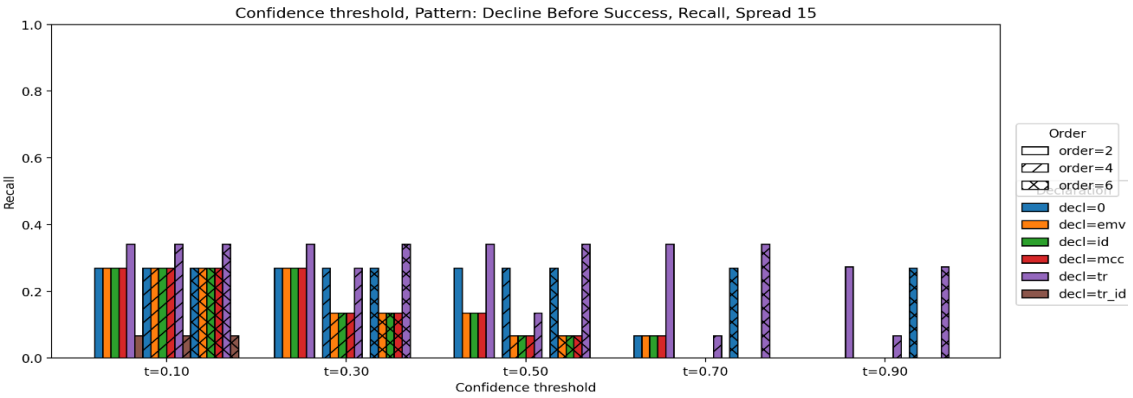

Figure 5. 32 Recall per Confidence Thresholds for Pattern Decline Before Success and Spread 15

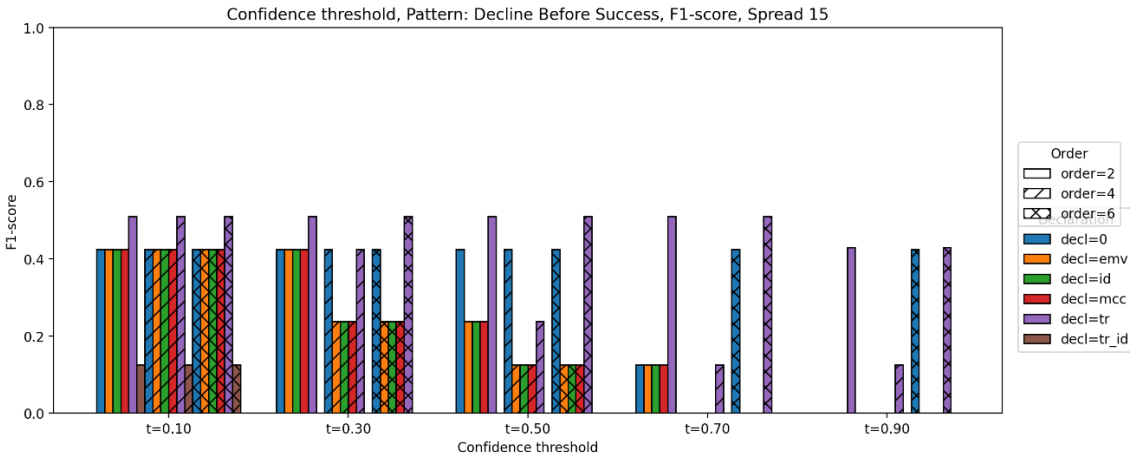

Figure 5. 33 F-1 Score per Confidence Thresholds for Pattern Decline Before Success and Spread 15

The figures (Figure 5. 28, Figure 5. 29, Figure 5. 30, Figure 5. 31, Figure 5. 32, Figure 5. 33) report precision, recall, and F1 as functions of the confidence threshold for two forecasting spreads (6 and 15). Bar hatching encodes the model order (2, 4, 6) and color encodes the declaration. Across all settings, precision is essentially saturated (≈1.0) whenever the model fires, the curves are therefore driven by recall, and F1 largely mirrors recall.

For spread = 6, the model concentrates probability near the end of each episode, so many windows exceed moderate thresholds. At t = 0.10–0.30 the generic setups (decl=0, emv, id, mcc) reach recall around ~0.65–0.68 and F1 around ~0.80, while decl=tr (TransactionResponse/Diff) is highest, sustaining recall ≳0.80 and F1 ≳0.90 across orders. At t = 0.50 the generic declarations drop sharply (recall ≈0.30 and F1 ≈0.25–0.50 depending on order), whereas decl=tr still holds ≈0.80–0.85 recall/F1. At t = 0.70–0.90 most declarations collapse to zero (no windows cross the threshold), but decl=tr with higher order still produces non-zero recall (≈0.80 at t=0.90 in your plot), hence strong F1. This shows both the benefit of using the response signal in the alphabet and the value of higher order when thresholds are stringent.

For spread = 15, the task is harder because probability mass is spread over far more offsets. At t = 0.10–0.30 generic declarations sit near recall ≈0.26–0.27 (F1 ≈0.42), while decl=tr is consistently higher (recall ≈0.33–0.35, F1 ≈0.50). At t = 0.50–0.70 those generic setups approach zero recall/F1, and only decl=tr maintains usable recall (≈0.50 at t=0.50 and ≈0.33 at t=0.70). At t = 0.90 virtually everything is zero except decl=tr (small but non-zero recall), which again tracks into the top F1. The contrast with spread=6 highlights the “dilution” effect of forecasting further ahead: each individual window gets a smaller probability slice, so fewer clear the higher thresholds.

Across both spreads we observe a non-monotonic dependence of F1 on model order. Contrary to the usual expectation that higher order should yield stronger probabilities, the F1 curves (e.g., spread-6 with decl=0 at t=0.7, and spread-15 with decl ∈ {0, tr}) often show :

- high at order=2 → low at order=4 → high at order=6

This fluctuation arises because order = 4 fragments the history into many rare contexts, making probability estimates under-confident (pushed toward ~0.5 by

smoothing) so few windows cross mid/high threshold, by contrast, order = 2 has reliable counts for short contexts, and order = 6 aligns with the full motif length, restoring a sharp end-of-episode probability peak. The effect is amplified by thresholding: small calibration shifts around 0.5–0.7 can flip many windows from positive to negative, producing the observed dip at intermediate order.

Comparing declarations, the winner is clear, decl=tr dominates across spreads and thresholds because it preserves the pattern's core signal (the decline→success transition) while keeping the alphabet compact. Adding high-cardinality fields (e.g., _id) fragments the alphabet, explodes minterms, and hurts evaluation metrics. The other single-field declarations (emv, id, mcc) behave close to baseline or slightly worse; they don't add predictive lift and can reduce it at higher thresholds. Finally, order matters mainly at higher thresholds: larger orders sharpen the end-of-episode peak so some windows cross t = 0.7–0.9 (visible for spread=6, weak for spread=15 unless the declaration is tr).

Lastly, the following figures are visual representations of metrics extracted from wayeb related to the pattern "Card Limit" by the declaring as extra predicates:

- As individual, "Transaction Response", decl = 1
- As combination, "Transaction Response" and "Cnp", decl = 2

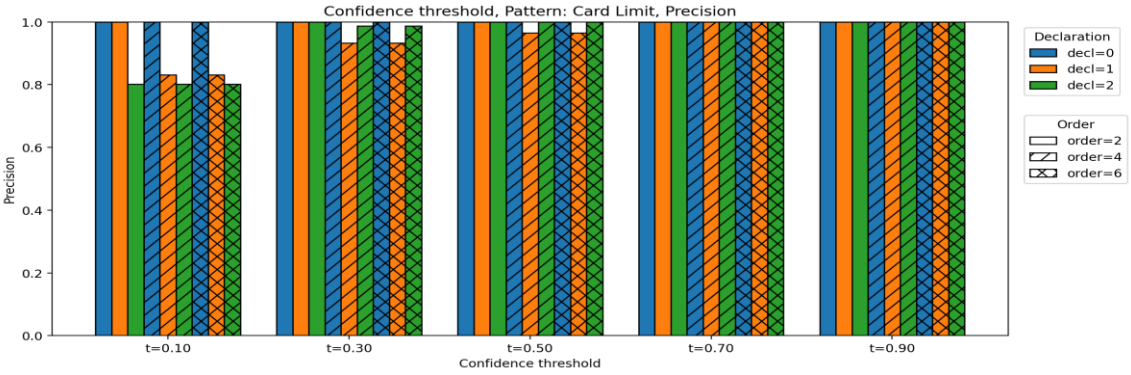

Figure 5. 34 Precision per Confidence Thresholds for Pattern Card Limit and Spread 7

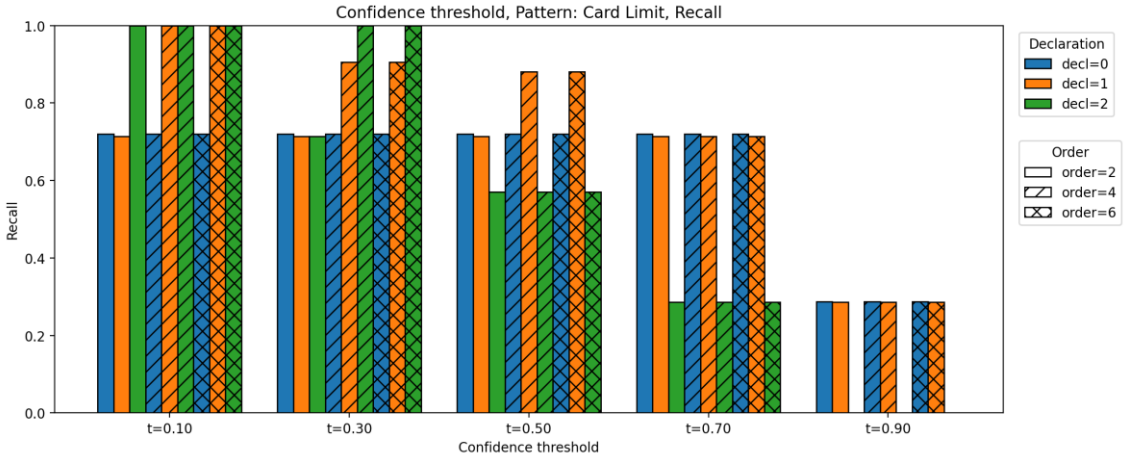

Figure 5. 35 Recall per Confidence Thresholds for Pattern Card Limit and Spread 7

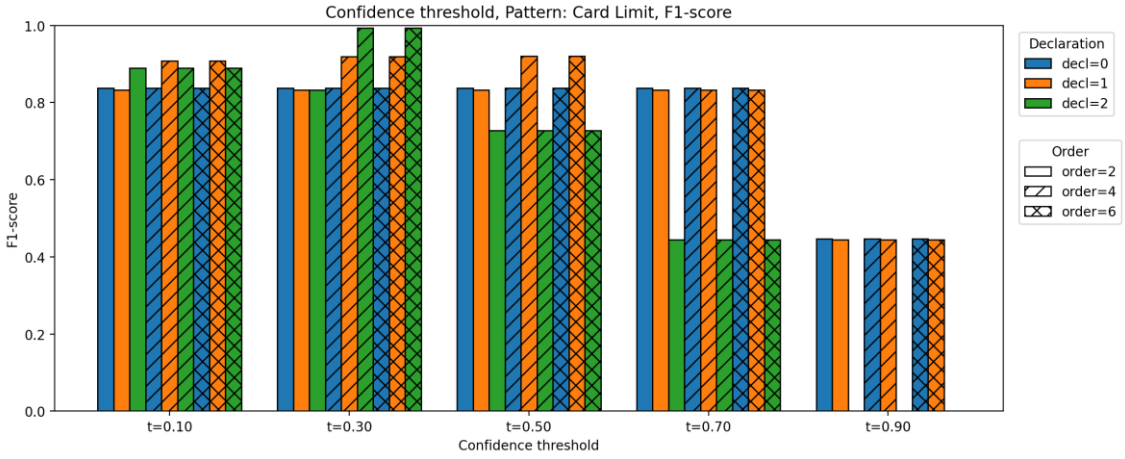

Figure 5. 36 F-1 Score per Confidence Thresholds for Pattern Card Limit and Spread 7

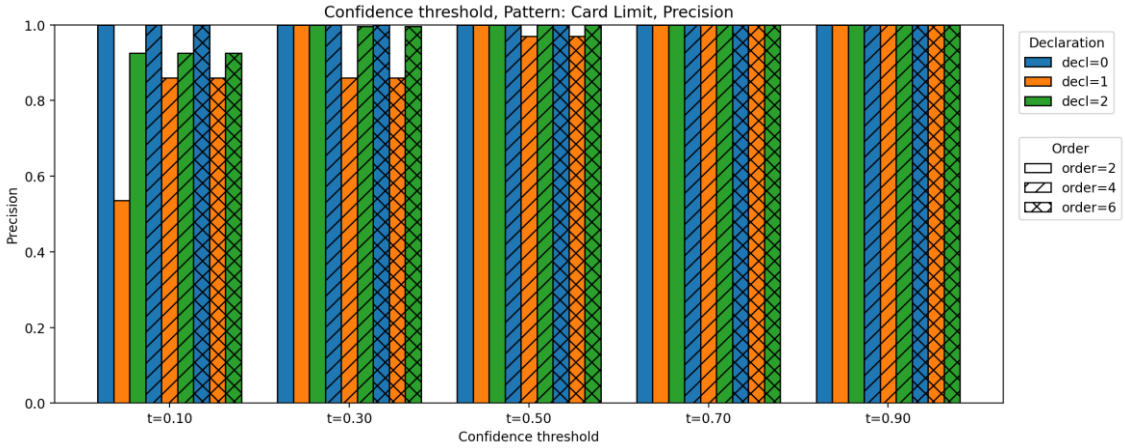

Figure 5. 37 Precision per Confidence Thresholds for Pattern Card Limit and Spread 15

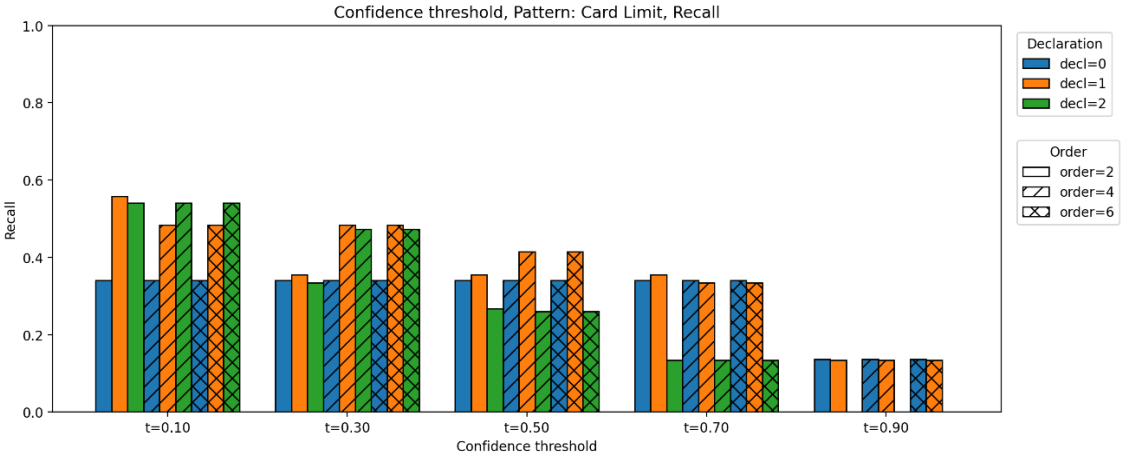

Figure 5. 38 Recall per Confidence Thresholds for Pattern Card Limit and Spread 15

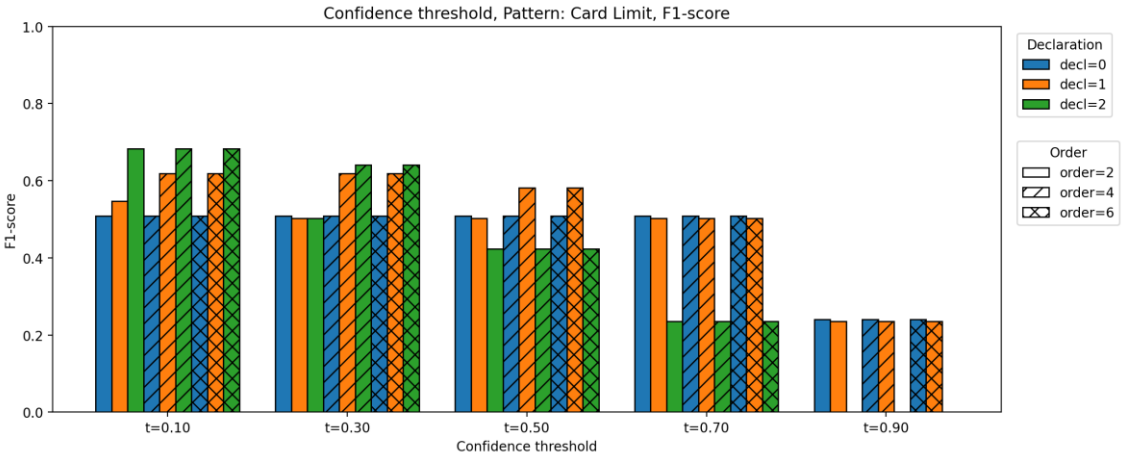

Figure 5. 39 F-1 Score per Confidence Thresholds for Pattern Card Limit and Spread 15

The figures (Figure 5. 34, Figure 5. 35, Figure 5. 36, Figure 5. 37, Figure 5. 38, Figure 5. 39) report precision, recall, and F1 as functions of the confidence threshold for two forecasting spreads (7 and 15). Bar hatching encodes the model order (2/4/6) and color the declaration. As in previous experiments, precision is essentially saturated (≈1.0) whenever the forecaster fires, the curves are therefore recall-driven, and F1 mostly mirrors recall.

For "Spread = 7" and with a 6-event SRE the model must see the first fraudulent event before probabilities ramp up, so the theoretical target is about 5/7 ≈ 0.71 recall. But the dataset's episodes last 10 events, so a 7-step window often overlaps all six SRE positions plus part of the tail (events 7–10). Practically, this yields ≈6/7 ≈ 0.86 recall, and sometimes even 7/7 when the window sits entirely inside the episode. Specifically:

- At t = 0.30 with decl = 2, the model attains recall = 1.00, precision = 0.987, and F1 = 0.993, indicating zero false negatives and very few false positives—an essentially ideal operating point for this task. But as the

threshold rises (t > 0.5) metrics are declined which indicates that probabilities are not strong as decl = 0 and 1.

- At decl = 0 (no declarations). Although the posterior probabilities for windows classified as positive remain strong across thresholds, the model does not achieve the same F1 as when informative attributes are declared (per feature-selection), confirming that incorporating those signals yields a measurable performance gain.
- At decl = 1 we can observe a little bit of improvement as the "order" rises and its strong till threshold = 0.7 specifically at "order" = 4 and 6.
- At t = 0.70–0.90, most variants thin out, decl=1 with higher order still produces non-zero recall, confirming the value of making the response signal explicit and of using larger order when thresholds are strict.

At t = 0.10, with decl = 1 or decl = 2, the model attains recall = 1.00 (no false negatives) but precision ≈ 0.80. This means the forecaster correctly flags all true frauds, yet also labels some additional windows as positive that are scored as false positives. The cause is the scoring mismatch between the data and the SRE: the dataset encodes a 10-event episode, while the SRE evaluates a 6-event motif. The first four "run-up" events come from the same distributional regime as the SRE tail, so the model assigns them high posterior probability and sometimes predicts them as positives. Under the 6-event SRE, those early hits are counted as FPs, even though they are semantically on-pattern (useful early warnings within the same fraud episode). Thus, the precision drop reflects evaluation granularity rather than model confusion. Operationally, this behavior is realistic and often desirable: detectors can reliably identify the tail where behavior deviates most from baseline, even if the exact episode boundary cannot be known a priori.

For Spread = 15 with a 6-event SRE, each fraud episode offers at most 6 "correct" offsets inside a 15-step forecasting window. Even with perfect localization, the ceiling on recall is ≈ 6/15 ≈ 0.40 (the forecaster can make up to 15 attempts per episode but only 6 time positions are counted as correct under the SRE). Because the dataset encodes 10-event Card-Limit episodes, the window frequently overlaps the SRE tail plus earlier run-up events, but those extra positions are not counted as correct by the 6-event SRE. This structural cap explains why recall and F1 are markedly lower than in spread = 7, while precision stays high once the model fires.

Concretely from the figures:

- t = 0.10–0.30. This is the only regime where recall gets close to the structural limit.
  – decl = 2 is usually best, reaching recall ≈ 0.5 with precision ≈ 0.9, so F1 ≈ 0.67.
  – decl = 1 follows closely and consistently improves over decl = 0, confirming the value of making the response signal explicit.
- t = 0.50. Recall drops for decl = 0 and decl = 2, while decl = 1 tends to degrade more gracefully, retaining the largest non-zero recall and therefore the top F1 at this threshold.
- t = 0.70–0.90. Most variants thin out to near-zero recall/F1 (few windows cross such strict cuts). Small non-zero bars appear mainly for decl = 1 with higher order, indicating that a longer memory can occasionally push the end-of-episode probability spike above very high thresholds—but the gains are modest and intermittent.

As with spread = 7, order has small, non-monotonic effects: increasing from

2→4→6 sometimes recovers a little recall at high thresholds, but there is no "always higher is better" trend. Under spread = 15, the spread itself dominates—probability mass is diluted over many offsets, so fewer windows exceed the threshold regardless of order. The spread = 15 configuration models a more realistic operational setting in which the first onset of a fraud episode is unknown. By forecasting over a long window and evaluating against a "tail" SRE motif (the distinctive end segment of the episode), the system focuses on the portion of behavior that is most clearly out of the genuine transaction distribution. In practice, this setup treats the tail as a high-confidence signature of fraud while acknowledging uncertainty about the exact episode start.

5.4 Integrating Feature Importance with Forecasting

This section synthesizes the feature selection outcomes of Section 5.2 with the automata-based forecasting results of Section 5.3. In short, the attributes identified as most informative in 5.2—when translated into SRE declarations—directly shaped the forecasting alphabet and, consequently, the model's probability calibration, recall/precision trade-offs, and sensitivity to order and spread. The figures in 5.3 show that declaring the right signals (e.g., TransactionResponse and, where useful, cnp) improves recall and F1 at practical thresholds, while superfluous or high-cardinality additions fragment the alphabet and depress performance. We therefore treat feature selection not as a standalone step, but as a design lever for the forecaster's symbolic interface, guiding which variables to expose in the alphabet and at what granularity.

For the "Merchant Mcc Diff" pattern, feature-selection analyses, both per method and in aggregate, highlighted MerchantMcc and amount as the most informative attributes. Those attributes according to the generator were indeed informative and followed a distribution in the fraud pattern. Yet, the forecaster's results did not improve when these variables were declared. The reason is structural, the generator instantiates this pattern in a very specific, highly separable way relative to the genuine distribution, so the core signal is already captured by the Diff trajectory itself. Adding MerchantMcc and amount to the alphabet expanded the number of symbols (minterms) and injected contextual variance, thinning the evidence per symbol and blurring the decision boundary. In practice, the declarations contributed noise rather than information, making the SRE less distinct and weakening detection.

For the Decline Before Success pattern, feature-selection analyses identified TransactionResponse, TerminalEmv, TerminalId, and MerchantMcc as the most informative attributes. We therefore evaluated declarations that introduced each attribute individually, as well as a joint declaration of TransactionResponse + TerminalEmv. This aligns with the generator's ground truth, in which these fields assume specific values within each fraudulent episode—confirming that FS was on target. In forecasting, the strongest and most consistent lift came from declaring TransactionResponse. It increased recall, capturing one additional fraudulent episode relative to other cases, and yielded better-calibrated, higher probabilities at stricter confidence thresholds. Declarations involving the other attributes offered smaller and less stable gains, reflecting their more contextual

role and the tendency of additional symbols to fragment the alphabet and dilute evidence per context.

For the Card Limit pattern, as we discussed above, the setup is intentionally asymmetric. The generator instantiates a 10-event episode, while the forecaster evaluates a 6-event "tail" SRE. This mirrors realistic operation—one rarely knows when a fraud episode begins—so the question is whether the model can raise actionable alerts before the sequence fully aligns with the SRE tail. Feature-selection singled out TransactionResponse and cnp as the most informative attributes, in agreement with the generator's ground truth. Declaring these signals substantially improved forecasting: introducing TransactionResponse yielded the most consistent gains in recall and calibration across thresholds, and augmenting it with cnp further enhanced early-warning behavior—occasionally triggering on the first events of an episode that lie outside the 6-event SRE. These anticipatory detections are scored as false positives under the strict SRE evaluation, but they are semantically on-pattern within the full 10-event episode and thus valuable in practice.

To sum up, the results in this chapter show that translating feature selection into SRE declarations can either sharpen or fragment the forecasting alphabet. For Merchant Mcc Diff, extra declarations added noise and weakened detection; for Decline Before Success, declaring TransactionResponse was decisively beneficial; and for Card Limit, TransactionResponse (and, selectively, cnp) improved early warning, with the 10-vs-6 episode mismatch explaining precision dips at low thresholds. Across spreads and orders, the strongest regimes share three traits: a compact, causal alphabet, moderate thresholds (≈0.30–0.50), and order roughly matched to the pattern's effective length. Overly rich or high-cardinality declarations inflate minterms, thin counts per context, and destabilize calibration. These insights motivate practical design rules—what to expose and at what granularity—and set the stage for the next chapter, where we distill general conclusions and outline directions for future work.

6 Conclusion and Future Work

This thesis examined how feature selection can be used as an alphabet–design instrument for automata-based complex event forecasting. By translating the most informative attributes into SRE declarations before determinization, we shaped the symbolic space the forecaster learns on—controlling minterms, stabilizing probability calibration, and ultimately affecting recall/precision trade-offs across model order and forecasting spread. In this framing, feature selection is not a preprocessing afterthought but a co-design lever that determines what the model "sees" and at what granularity.

Empirically, three patterns highlighted different roles for declarations. For "Merchant Mcc Diff", although feature selection ranked MerchantMcc and amount highly, declaring them did not help the generator instantiated a very specific, separable trajectory, so the Diff signal already captured what the forecaster needed, and extra predicates only fragmented the alphabet and weakened performance. For "Decline Before Success", declaring "Transaction Response" aligned the alphabet with the decline→success mechanism and consistently improved recall and F1, especially at moderate and high thresholds, adding other fields yielded smaller and less stable gains, reflecting their contextual rather than causal role. For "Card Limit", the dataset encoded a 10-event episode while the SRE evaluated a 6-event tail, a deliberate asymmetry that mirrors realistic uncertainty about onset. Declaring "Transaction Response" and, at lenient thresholds, cnp sharpened the end-of-episode spike and enhanced early-warning behavior. Some anticipatory detections were scored as false positives under the strict 6-event evaluator, but they were semantically on-pattern within the longer episode and therefore operationally useful.

Across patterns, order and spread mattered in consistent ways. Increasing order helped when it matched the effective motif length, but gains were non-monotonic at intermediate orders due to data sparsity and alphabet fragmentation. Enlarging spread diluted probability mass over more offsets, effectively capping recall per episode and making high thresholds harder to satisfy. In practice, the most balanced regimes combined a compact, causal alphabet with thresholds around 0.10–0.30 and an order sized to the pattern's temporal footprint, very high thresholds were only viable when declarations exposed a strong driver.

There are limitations to acknowledge. Results were obtained on synthetic streams with planted patterns, while this affords controlled experimentation, it cannot exhaust the variability, drift, and adversarial behavior of real deployments. Evaluation asymmetries, most notably the 10-vs-6 episode in Card Limit, provided a strict lower bound on measured precision but understated early-warning value. Finally, declaration granularity (binning choices, "other" buckets for rare categories) and the specific learning pipeline (DSFA + PST with smoothing/back-off) influence calibration. Alternative learners or coarsenings

may shift the exact trade-offs while preserving the qualitative lessons.

From these findings emerge concrete design guidelines framed as prose rather than prescriptions. Start small and causal: declare attributes that encode the progression of the pattern and avoid high-cardinality context unless it demonstrably lifts calibration. Tune order to the motif length rather than assuming that more memory is always better. Choose operating thresholds pragmatically, mid-range values typically maximize F1 by preserving recall without sacrificing already-high precision. Align evaluation with intent, when forecasts are meant as early warnings, adopt a scoring protocol that credits timely on-pattern alerts within a small time-to-go tolerance alongside strict tail matching.

For future work, one line is learnable declarations — searching the predicate set and binning under a complexity budget so the alphabet remains compact while predictive. Another is adaptive coarsening that merges rarely used symbols online to counteract fragmentation and drift. A third is robust evaluation for early warnings, complementing strict SRE scoring with time-to-event metrics that quantify operational value. Beyond this, validating on real streams with distribution shift, scarce labels, and adaptive adversaries, jointly training neural selectors against a downstream forecasting loss, and making operation cost-aware (latency, state budget, throughput) would turn the co-design principle into a deployable recipe.

In sum, feature selection has a tangible, controllable impact on automata-based complex event forecasting once its outcomes are expressed as SRE declarations. By curating a small, causal alphabet, we improved recall and F1 at practical thresholds while keeping the machinery interpretable and efficient. The broader lesson is methodological: design the alphabet you want to learn on, and let selection, coarsening, and evaluation co-evolve with the forecaster.

7 References

[1] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, Minos Garofalakis, «Complex Event Recognition in the Big Data Era: A Survey,» *VLDB Journal,* τόμ. 29, pp. 313 - 352, 2020.

[2] Sergio Ramírez-Gallego, Bartosz Krawczyk, Salvador García, Michał Wozniak, Francisco Herrera, «A survey on Data Preprocessing for Data Stream Mining: Current Status and Future Directions,» *Elsevier,* τόμ. 239, pp. 39 - 57, 2017.

[3] Herik, Laurens van der Maaten, Eric O. Postma, Jaap van den, «Dimensionality Reduction: A Comparative Review,» 2008.

[4] Jun Yan, Benyu Zhang, Ning Liu, Shuicheng Yan, Qiansheng Cheng, W. Fan, «Effective and efficient dimensionality reduction for large-scale and streaming data preprocessing,» *IEEE Transactions on Knowledge and Data Engineering,* τόμ. 18, αρ. 3, pp. 320 - 333, March 2006.

[5] Benyamin Ghojogh, Maria N. Samad, Sayema Asif Mashhadi, Tania Kapoor, Wahab Ali, Fakhri Karray, Mark Crowley, «Feature Selection and Feature Extraction in Pattern Analysis: A Literature Review,» 2019.

[6] Weikuan Jia, Meili Sun, Jian Lian, Sujuan Hou, «Feature dimensionality reduction: a review,» *Complex & Intelligent Systems,* τόμ. 8, pp. 2663 - 2693, 2022.

[7] M. Sipser, «Part One: Automata and Languages,» σε *Introduction to the Theory of Computation*, 1997, pp. 29 - 154.

[8] W. Thomas, «Languages, Automata, and Logic,» 1996.

[9] M. O. Rabin, «Probabilistic automata,» *Information and Control,* τόμ. 6, αρ. 3, pp. 230 - 245, 1963.

[10] Sagarkumar S. Badhiye, P. N. Chatur, «Range Automata for Alphabetic Time Series Dimensionality Reduction,» σε *International Conference on Secure Cyber Computing and Communication*, 2019.

[11] Elias Alevizos, Alexander Artikis, Georgios Paliouras, «Complex Event Forecasting with Prediction Suffix Trees,» *VLBD Journal,* 2021.

[12] Cugola, Gianpaolo and Margara, Alessandro, «Processing flows of information: From data stream to complex event processing,» *ACM Comput. Surv.,* τόμ. 44, 2012.

[13] Kostas Patroumpas, Elias Alevizos, Alexander Artikis, Marios Vodas, Nikos Pelekis, Yannis Theodoridis, «Online event recognition from moving vessel trajectories,» τόμ. 21, p. 389–427, 2016.

[14] Howar F., Steffen B., Merten M., «Automata Learning with Automated Alphabet,» σε *Verification, Model Checking, and Abstract Interpretation*, 2011.

[15] Amaldev Manuel, R.Ramanujam, «Automata over Infinite Alphabets,» σε *Modern Applications of Automata Theory*, 2012, pp. 529 - 553.

[16] Michal Šedý, Lukáš Holík, «Automata Size Reduction by Procedure Finding,» 2024.

[17] D'Antoni, Loris & Veanes, Margus, «The Power of Symbolic Automata and Transducers,» σε *International Conference on Computer Aided Verification*, 2017.

[18] M. Davis, Markov Models & Optimization, 1993.

[19] S. R. Eddy, «Hidden Markov models,» *Current Opinion in Structural Biology,* τόμ. 6, αρ. 3, pp. 361 - 365, 1996.

[20] L. E. -. Fosler, «Markov Models and Hidden Markov Models: A Brief Tutorial,» *INTERNATIONAL COMPUTER SCIENCE INSTITUTE,* 1998.

[21] Sandeep, «How Deep Learning is Revolutionizing Online Transaction Fraud Detection,» 2025. [Ηλεκτρονικό]. Available: https://opencv.org/blog/online-transaction-fraud-detection-using-deep-learning/.

[22] M. Datawarehouse, «SECURE PAYMENTS WITH AI FRAUD PREVENTION SOLUTIONS,» 2022. [Ηλεκτρονικό]. Available: https://www.mastercard.com/gateway/payment-solutions/secure-payments/fraud-protection.html.

[23] Dawei Cheng, Yao Zou, Sheng Xiang, Changjun Jiang, «Graph neural networks for financial fraud detection: a review,» *Frontiers of Computer Science,* τόμ. 19, 2025.

[24] Yue Tian, Guanjun Liu, Jiacun Wang, Mengchu Zhou, «Transaction Fraud Detection via an Adaptive Graph Neural Network,» 2023.

[25] Xuegang Hu, Peng Zhou, Peipei Li, Jing Wang & Xindong Wu, «A survey on online feature selection with streaming features,» *Frontiers of Computer Science,* τόμ. 12, pp. 479 - 493, 2018.

[26] Annie Anak Joseph, Takaomi Tokumoto, Seiichi Ozawa, «Online feature extraction based on accelerated kernel principal,» *Evolving Systems,* τόμ. 7, pp. 15 - 27, 2016.

[27] L.C. Molina, L. Belanche, A. Nebot, «Feature Selection Algorithms: A Survey and Experimental Evaluation,» σε *IEEE International Conference on Data Mining (ICDM)*, 2002.

[28] «Information Gain and Mutual Information for Machine Learning,» 2025. [Ηλεκτρονικό]. Available: https://www.geeksforgeeks.org/machine-learning/information-gain-and-mutual-information-for-machine-learning/.

[29] «mutual_info_classif,» 2016. [Ηλεκτρονικό]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html.

[30] «Pearson Correlation Coefficient,» 2025. [Ηλεκτρονικό]. Available: https://www.geeksforgeeks.org/maths/pearson-correlation-coefficient/.

[31] «r_regression,» 2016. [Ηλεκτρονικό]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.r_regression.html.

[32] «Chi-Square Test,» 2025. [Ηλεκτρονικό]. Available: https://www.geeksforgeeks.org/maths/chi-square-test/.

[33] «chi2,» 2016. [Ηλεκτρονικό]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html.

[34] «Autoencoders in Machine Learning,» 2025. [Ηλεκτρονικό]. Available: https://www.geeksforgeeks.org/machine-learning/auto-encoders/.

[35] A. Tate, «Using Autoencoders for Feature Selection,» 2023. [Ηλεκτρονικό]. Available: https://hex.tech/blog/autoencoders-for-feature-selection/.

[36] Shuyang Wang, Zhengming Ding, Yun Fu, «Feature Selection Guided Auto-Encoder,» σε *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[37] Wael Hassanieh, Abdallah Chehade, «Selective Deep Autoencoder for Unsupervised Feature Selection,» σε *TheThirty-Eighth AAAI Conference on Artificial Intelligence*, 2024.

[38] Dhananjay Tomar, Yamuna Prasad, Manish K. Thakur, K. K. Biswas, «Feature Selection Using Autoencoders,» σε *International Conference on Machine Learning and Data Science (MLDS)*, 2017.

[39] Tingting Chen, Xueping Liu, Bizhong Xia, Wei Wang, Yongzhi Lai, «Unsupervised Anomaly Detection of Industrial Robots using Sliding-Window Convolutional Variational Autoencoder,» *IEEE Access,* τόμ. 8, pp. 47072 - 47081, 2020.

[40] Alexey Chernikov, Chang Wei Tan, Pablo Montero-Manso, Christoph Bergmeir, «FRANS: Automatic Feature Extraction for Time Series Forecasting,» 2022.

[41] Elias Alevizos, Alexander Artikis, Georgios Paliouras, «Wayeb: a Tool for Complex Event Forecasting,» σε *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning,*, 2018.

[42] B. C. Ross, «Mutual Information between Discrete and Continuous Data Sets,» *PLoS ONE,* τόμ. 9, αρ. 2, 2014.