ForeSPECT: A Model-Driven Framework for Validation and Traceability in Forecasting Systems

1st Rijul Saini
NAV CANADA
Ottawa, Canada
Rijul.Saini@navcanada.ca

***Abstract*—Organizations increasingly rely on forecasting systems to anticipate future conditions and inform their strategic planning. However, current practices for specifications of these systems are scattered across workflows. Moreover, these specifications are either loosely defined or tied to data representation formats that lack domain awareness and offer only superficial validation. These limitations make it difficult to ensure correctness, enforce compliance, and trace qualitative adjustments across forecasting workflows. To address these challenges, we propose ForeSPECT, a model-driven framework for Forecasting with Semantic Provenance, Evaluation, Compliance, and Traceability. The framework introduces a metamodel that serves as the foundation for semantic validation and adjustments traceability, enabling early detection of domain-specific inconsistencies that conventional schema-based rules often miss. Our approach shows promise based on evaluation with nine unseen real-world datasets, achieving 77.7% mapping coverage between the metamodel and actual time-series record entities. It further demonstrates better performance in detecting errors earlier than pipeline-based methods, while ensuring 100% forward and 91% backward traceability of adjustments.**

***Index Terms*—Forecasting, Scenarios, Analysis, Domain Specific Language, Modelling**

I. Introduction

Organizations commonly use scenarios to explore alternatives and develop strategies. These scenarios are engaging narratives that represent plausible combinations of future events, which are often mutually exclusive but not necessarily exhaustive [1]. With the analysis of trends and patterns using data from the past and present, forecasting equips individuals with the insights needed to navigate complexities, anticipate challenges, minimize risks, and take advantage of opportunities in a landscape surrounded by uncertainty [2]. Scenario-based analysis in forecasting projects requires that the underlying system responsible for driving the forecasting activities be flexible and coherent to address evolving conditions over time. These projects typically involve collaboration among diverse stakeholders such as data scientists, engineers, and domain experts. These stakeholders collectively document and interpret changing scenarios to inform decision-making. To facilitate this, various formats such as XMI, JSON, and YAML are used for tasks like data serialization, configuration management, and model exchange across forecasting workflows. However, these data formats often lack the domain-specific context which can weaken the validation process in forecasting systems. Moreover, forecasting adjustments are typically expressed by different stakeholders manually in different channels like spreadsheets, adhoc conversations, and emails with hidden assumptions making it difficult to align these adjustments with scenario specifications.

In this paper, we propose **F**ore**SPECT**, a model-driven framework for **F**orecasting with **S**emantic **P**rovenance, **E**valuation, **C**ompliance, and **T**raceability. The semantic backbone of our proposed framework is the metamodel that captures the core concepts, relationships, and constraints, serving as the foundation pillar for the semantic validation and traceability of adjustments. While schema-based approaches primarily enforce syntactic correctness, our model-driven framework aims to detect semantic inconsistencies in the early stages to enforce compliance across forecasting workflows. Moreover, our proposed approach aims to establish semantic provenance to facilitate the traceability of qualitative adjustments and reduce the risk of hidden assumptions. In this paper, the main contributions are:

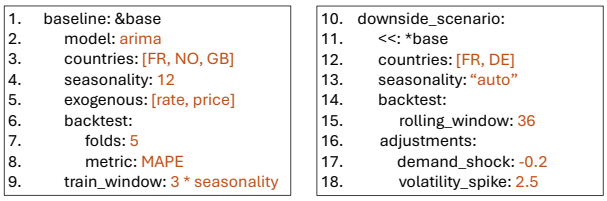
1) We identify the limitations of current specification formats in capturing domain semantics that limit their support for validation and traceability.
2) We propose a model-driven framework to enable semantic validation and traceability.
3) We present our metamodel that captures core concepts and relationships in scenario-based forecasting systems.
4) We instantiate the core components of our framework and build an automated pipeline for its evaluation.
5) We evaluate our approach using 9 real-world datasets where our approach achieves 77.7% mapping coverage to time-series data, an average speed-up of 0.7 seconds per error, and promising traceability results.

The rest of the paper is organized as follows: Section II highlights the limitations of YAML-based forecasting specifications and motivates our approach. Section III surveys related work. Section IV describes our proposed framework and presents the metamodel. Section V reports evaluation results. Finally, Section VI concludes with future work.

II. Motivation

Scenario-based analysis in forecasting often involves modelling and exploring alternative trends that differ from the base scenario, which could lead to significant changes in existing conditions [3]. This highlights the need for a coherent

2151-0849/25/$31.00 ©2025 IEEE
DOI 10.1109/ASEW67777.2025.00035



(a) Baseline Scenario (b) Downside Scenario

Fig. 1: Example of YAML-based Specification for Scenarios

and adaptive system with strong domain awareness to ensure semantic validation and traceability of decisions. Furthermore, adjustments at the domain level are often applied to forecasts produced by statistical or machine learning models to incorporate qualitative factors that quantitative data does not capture. However, existing specification formats which drive forecasting workflows with features such as data serialization, configuration management, and model exchange fall short in this regard, limiting their effectiveness for nuanced scenario modelling. In this section, we highlight current limitations using the YAML data format which has gained widespread adoption due to its simplicity, human readability, and ease of integration with modern tooling [4].

Missing Semantic Validation: As shown in Figure 1, defining baseline and downside scenarios using the YAML format introduces several challenges. For example, the country option "NO" in Line 3 (L3) of the baseline scenario is interpreted as Boolean *false* in YAML, leading to semantic ambiguity. Similarly, units for parameters such as seasonality (L4) and rolling_window (L15) are unspecified (they may represent different scales such as days and months). Moreover, it is difficult to enforce constraints such as the exogenous variable "Price" is only valid when the data frequency is daily (L5), perform strict type checks for fields accepting only string or integer (L13), or to enable computation within YAML (L9). Although YAML supports anchors (&) and aliases (*) for content reuse, it lacks object-oriented features such as inheritance and template-based overrides. For example, in the downside scenario, the backtest section silently overwrites the metric and folds fields. While it is possible to use separate anchor for the backtest to mitigate this, it reduces readability and makes the specification harder to maintain.

Missing Provenance: Domain-level qualitative adjustments are often communicated through spreadsheets or emails, embedding hidden assumptions that are difficult to capture and trace. This fragmentation complicates the task of establishing adjustments related provenance. For example, it is difficult to validate the bounds for volatility_spike factor (e.g., distinguishing between 250% and 2.5%) without close collaboration between domain experts and engineers (L18). Without provenance, it is challenging to audit the historical qualitative adjustments or reproduce these adjustments.

The above limitations result in superficial validation, leaving deeper inconsistencies and misalignments with domain logic remain undetected until the later stages of execution. This not only increases rework and debugging efforts, but also introduces the risk of critical errors surfacing during deployment or runtime. Current practices rely on external tools like Pydantic [5] for validation, but these tools also lack domain awareness, making them unsuitable for cross-file validation, model transformations, and traceability. These challenges identify the need for an approach that facilitates semantic provenance and validation, ensuring that scenario-based modelling remains reliable and transparent.

III. Related Work

We organize the related work into two areas: (1) Validation in Forecasting and (2) Traceability in Forecasting.

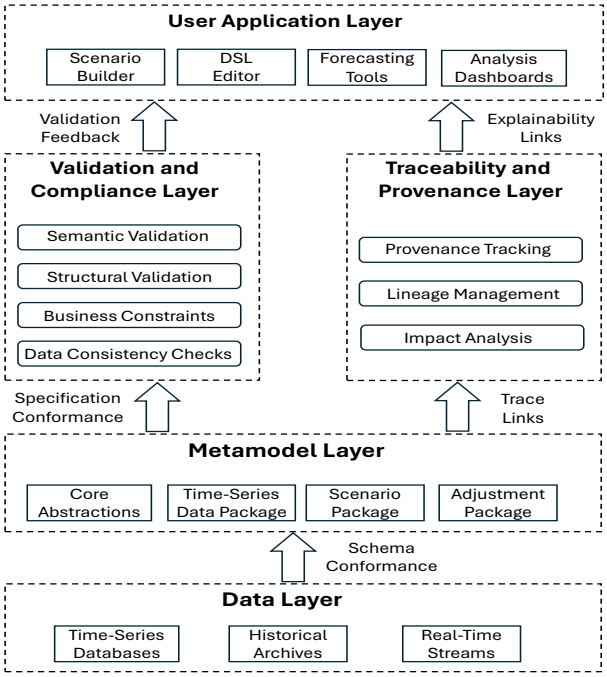
A. Validation in Forecasting: Traditional approaches to forecasting validation primarily emphasize accuracy assessment and model comparison [6]–[8]. Hyndman et al. outline practices involving residual diagnostics, accuracy metrics, and cross-validation of time-series, which remain the standard reference for evaluating both statistical and machine learning forecasting methods [9]. More recent work by Hewamalage et al. highlights how challenging time-series characteristics such as non-normality and non-stationarity introduce pitfalls in forecast evaluation, underscoring the limitations of classical validation techniques [10]. The increasing demand for large-scale forecasting has spurred interest in automated pipeline-based workflows [11]. These pipelines rely on specifications that define what and how forecasts should be executed, often in collaborative environments. This highlights the need for unified frameworks to support semantic-based validation. Kansab performs a literature review to examine state-of-the-art machine learning pipelines and emphasizes the need for dynamic and domain-aware pipeline designs that align with the evolving demands of Software Engineering [12]. However, to the best of our knowledge, no existing approaches provide semantic validation beyond basic schema-level checks for forecasting pipelines or workflows. To address this gap, we propose a model-driven framework that ensures forecasting workflows remain correct, valid, and compliant with specifications.

Traceability in Forecasting: Traceability has largely been discussed in the context of reproducibility and auditability of analytical workflows in the Data Science space [13], [14]. Early work in scientific workflow systems highlight the importance of provenance to record and track computational steps, thereby enabling reproducibility and explanation of results [15], [16]. However, traceability remains unexplored in the forecasting space, with most approaches emphasizing data lineage and version control rather than semantic associations between forecasts, assumptions, and adjustments. Recent studies on machine learning and time-series forecasting pipelines have stressed the necessity of tracking data transformations, hyperparameter settings, and model versions to enable transparent decision-making [17], [18]. However, these solutions typically capture low-level execution traces, offering little support for connecting forecasts back to domain-specific assumptions or scenario adjustments. As scenario-based forecasting becomes increasingly relevant for decision

support, there is a growing need for frameworks that establish provenance to include semantic links between inputs, scenarios, adjustments, and final outputs. To the best of our knowledge, no framework exists that enables semantic-based provenance to enable traceability in forecasting workflows. Our work addresses this gap by using a model-driven approach to capture and reason about such traceability links.

IV. ForeSPECT: Our Proposed Framework

Due to the growing complexity of forecasting systems and underlying workflows, it is non-trivial to design scenario-based forecasting systems that are both compliant and traceable. To address this requirement and the fundamental challenges outlined in Section II, we present our proposed framework in this section. As shown in Figure 2, our framework is composed of five layers - *User Application*, *Validation and Compliance*, *Traceability and Provenance*, *Metamodel*, and *Data Layers*.



Framework.pdf

Fig. 2: ForeSPECT: Our Proposed Framework

A. Data Layer

To provide unified access to diverse data sources, the *Data Layer* integrates heterogeneous inputs that support various forecasting tasks. These sources capture both static and dynamic aspects of the system to enable adaptive and real-time forecasting. The processed data conforms to the metamodel, with schema enforcement facilitated by data specification formats (e.g., YAML). These specifications act as intermediaries between the metamodel and the data layer, ensuring schema consistency and interoperability across forecasting workflows.

1) **Time-Series Databases** are designed to handle a large volume of data and facilitate specialized tasks such as time-series data modeling, time-based aggregation, and downsampling. These databases use optimized data structures to enable fast ingestion, efficient querying of time ranges, and effective compression for large datasets.
2) **Real-Time Streams** provide continuous data feeds from sensors, APIs, or transactional systems. These streams enable forecasting systems to respond dynamically to outliers and adapt to changing conditions in real time.
3) **Historical Archives** store long-term data snapshots that are often used for training forecasting models, benchmarking, and performing retrospective analysis.

B. Metamodel Layer

The metamodel serves as the semantic backbone of the ForeSPECT framework, providing both formal structure and domain-aware semantics to support forecasting activities. It provides a coherent schema to align heterogeneous data, scenarios, and user interactions in the form of a specification. As shown in Figure 3, our metamodel is organized into four packages: *Core Abstractions*, *Time-Series Data Package*, *Scenario Package*, and *Adjustment Package*.

1) **Core Abstractions** introduces generic abstractions that recur across the metamodel. The *Entity* abstract-class provides attributes such as *id*, *name*, and *description*. The *Traceable* interface captures provenance information using trace identifiers (*traceID*), parent–child relationships, and depth level within the trace graph. The *Validatable* interface indicates that a specification can be subjected to semantic, structural, and business constraints. These abstractions ensure consistency across all domain-specific constructs.
2) **Time-Series Data Package** models the structure and semantics of input data used in forecasting tasks. The *Variable* class defines observed features, including their roles (`TARGET`, `PREDICTOR`, `EXOGENOUS`), types, and data quality attributes such as *hasGaps* and *gapPercentage*. The *TimeSeriesProfile* class encapsulates metadata such as *sourceSystem*, *granularity* (e.g., `MONTHLY` and `YEARLY`), and associated variables. *DataPoint* class stores timestamped values and quality indicators for anomalous or adjusted data points. The *SeriesType* class with options such as `UNIVARIATE` and `MULTIVARIATE` allows explicit representation of the data modelling context.
3) **Scenario Package** captures alternative forecasting experiments and their execution states. A *ForecastingModel* class specifies the modeling approach, scope (e.g., `LOCAL` and `GLOBAL`), method (e.g., `ARIMA` and `PROPHET`), and their associated parameters. The *Horizon* class distinguishes between training, validation, and testing periods. Forecast results are stored in *ForecastingResults*, which link to collections of *ForecastPoints* containing predicted values, bounds, and confidence levels. The *Scenario* class organizes and maintains attributes such as *owner*, *confidenceLevel*, and *status*. Dependencies between scenarios allow representation of hierarchical or sequential scenarios, supporting what-if analysis.

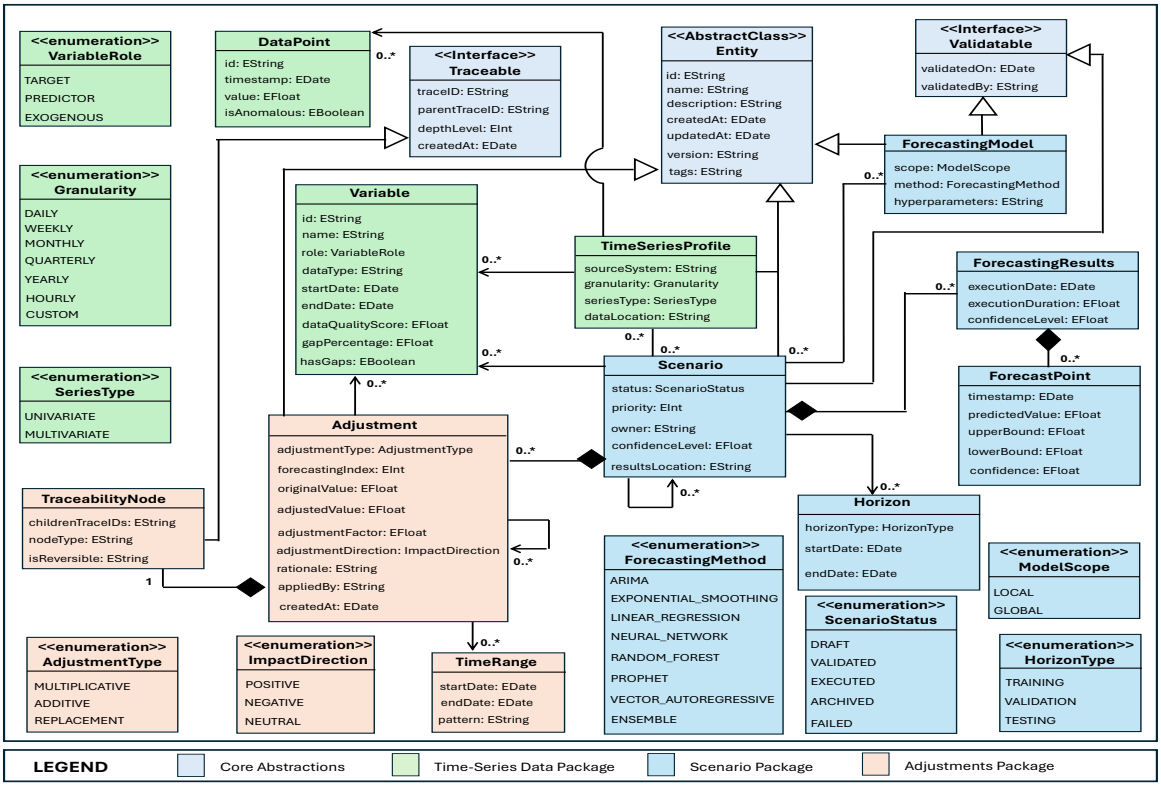


Fig. 3: Our Proposed Metamodel

4) **Adjustment Package** captures post-forecasting interventions on forecasts and their traceability. The *Adjustment* class specifies the type of change (`MULTIPLICATIVE`, `ADDITIVE`, and `REPLACEMENT`), its magnitude (*adjustmentFactor*), *rationale*, and the time range to which it applies. Adjustments are linked to a *TraceabilityNode*, enabling provenance tracking and traceability of these adjustments. The attribute *ImpactDirection* (`POSITIVE`, `NEGATIVE`, `NEUTRAL`) provides additional semantic context about the effect of adjustment. The *TimeRange* class specifies the temporal scope of the analysis. Dependencies between adjustments support reasoning over cascading or compound modifications.

The metamodel only includes those concepts that can be represented in specification formats (e.g., YAML, JSON). For example, metamodel includes the *DataPoint* class only to capture the information about anomalous or adjusted data points and not for every data point as they are usually stored in the *Data Layer*. In summary, the metamodel provides a unifying schema that integrates data, models, adjustments, and scenarios. By encoding both technical and semantic aspects, it facilitates schema conformance, ensures explainability, and enables systematic validation of forecasting specifications.

C. Validation and Compliance Layer

The role of Validation and Compliance Layer in our proposed framework is to ensure that the user activities in forecasting workflows are correct, valid and compliant to business constraints and objectives. While the metamodel defines the structure of scenario-based forecasting, this layer enforces domain-level rules to support specification conformance to the metamodel and safeguard against inconsistent data usage, and logically incorrect scenarios. This layer is organized into four main components: *Semantic Validation*, *Structural Validation*, *Business Constraints*, and *Data Consistency Checks*. These components provide a multi-level validation mechanism that goes beyond simple schema-based validation. In this paper, we instantiate all the components in this layer with a set of rules to evaluate our framework and discuss the evaluation results in Section V. Some rules may impact multiple components.

1) **Structural Validation** aims to provide fundamental blocks for ensuring that forecasting specifications are correctly defined. This component enforces schema-level correctness to ensure that the specifications can be parsed, stored, and processed without failure. Some structural checks include:
   a) Presence of required attributes (e.g., missing *id* in a *TimeSeriesProfile*).
   b) Correct data types (e.g., ensuring *variables* in a given *Scenario* is a list rather than a single string).
   c) Valid enumerations (e.g., rejecting invalid model types).
   d) Required sections (e.g., preventing removal of the entire *ForecastingModel* definition).
2) **Semantic Validation** component ensures that domain

classes and their relationships in our metamodel are logically consistent when instantiated. The main goal of this component is to identify errors that, while structurally valid, are semantically incorrect, as these can lead to misleading forecasts. Some semantic checks include:
   a) Valid date logic (e.g., preventing an end date that precedes the start date).
   b) Reasonable percentages (e.g., identifying when data *gapPercentage* is greater than 100% or some threshold specified by domain experts).
   c) Valid confidence ranges (e.g., *confidenceLevel* must fall within [0,1] or within a pre-defined range).
   d) Data quality safeguards (e.g., identifying negative or poor data quality scores).
3) **Business Constraints** introduces domain-specific constraints that reflect how scenarios should be modelled and forecasting models should be applied in practice. These constraints enforce that the chosen models, horizons, and configurations are aligned with the business use case and comply with the business strategy. In addition, these business constraints ensures that forecasts remain practical and interpretable. Some business constraints include:
   a) Model-data alignment (e.g., preventing the use of a univariate model on multivariate time-series data).
   b) Horizon feasibility (e.g., ensuring that forecast horizons are not longer than the historical or training horizons).
   c) Non-overlapping horizons (e.g., avoiding contradictory training, validation, and test time ranges).
4) **Data Consistency Checks** ensure coherence between linked components. These checks bridge the gap between semantic validation and business rules by addressing both logical correctness and domain alignment. In our work, we position them as a distinct component to emphasize the role of cross-entity validation in scenario-based forecasting systems. Some data consistency checks include:
   a) Reference integrity (e.g., avoiding references to non-existent variables in time-series).
   b) Granularity consistency (e.g., preventing mismatches between data series granularity and scenario horizons).

D. Traceability and Provenance Layer

The Traceability and Provenance Layer aims to provide visibility into the lifecycle of forecasting artifacts, ensuring that every transformation, adjustment, and dependency is stored and auditable. While the metamodel defines the structural representation of the forecasting entities, this layer ensures that their evolution over time is transparent, reproducible, and auditable. This is particularly critical for regulatory compliance, debugging, and impact assessment in complex forecasting workflows. The metamodel classes responsible for this layer include *Traceable*, *TraceabilityNode*, and *Adjustment*, which collectively provide the structural foundation for implementing the trace links. The layer is organized into three main components: *Provenance Tracking*, *Lineage Management*, and *Impact Analysis*. In this work, we instantiate *Provenance Tracking* and evaluate our framework in Section V.

1) **Provenance Tracking** captures the origin and transformation history of each forecasting *Adjustment* entity. It ensures that every adjustment is associated with metadata such as who performed the change (*appliedBy*), when it occurred (*createdAt*), and why it was applied (*rationale*). This component leverages the *Traceable* interface and *TraceabilityNode* class to maintain hierarchical trace chains, enabling reconstruction of the exact state of the system at any point in time. Key attributes include:
   a) *traceID* and *parentTraceID* for linking adjustment operations in a trace hierarchy.
   b) *depthLevel* for representing the nesting of changes.
   c) *createdAt* for the temporal ordering of events.
2) **Lineage Management** focuses on establishing and maintaining relationships between core constructs such as *Scenario* and *Adjustment*) across different stages of the workflow. It enables both upstream and downstream lineage queries. For example, *Lineage Management Component* may allow users to propagate the impact of an adjustment operation forward to dependent scenarios. In our metamodel, it is possible to achieve this lineage through references such as *dependencies* in the *Adjustment* class and child-parent relationships in *TraceabilityNode*. Common lineage operations may include:
   a) Identifying adjustments that contribute to a forecast.
   b) Mapping dependencies between scenarios and models.
   c) Using lineage graphs for audit and debugging purposes.
3) **Impact Analysis** provides support to assess the consequences of changes within forecasting workflows. By using the traceability graph and lineage metadata, this component aims to determine which forecasts, scenarios, or Key Performance Indicators (KPIs) are affected by a given adjustment or model update. This is non-trivial for risk assessment and what-if analysis. Impact analysis supports:
   a) Forward impact propagation (e.g., which forecasts may change if a variable is adjusted).
   b) Backward impact tracing (e.g., which inputs influence a specific forecast).

E. User Application Layer

Forecasting workflows typically require active collaboration among different stakeholders such as data scientists, engineers, and domain experts. The *User Application Layer* serves as the primary interaction point between these stakeholders and the underlying forecasting infrastructure. By collecting validation feedback and explainability links from the *Validation* and *TraceabilityLayers*, respectively, the *User Application Layer*, ensure that user activities are valid and interpretable. This layer abstracts the complexity of forecasting systems by providing intuitive interfaces for scenario specification and analysis. This layer may include the following components:

1) **Scenario Builder** offers interactive interfaces to build, modify, and compare scenarios. Its goal is to ensure semantic consistency through validation feedback.
2) **Domain-Specific Language (DSL)** editor allows different stakeholders to work on forecasting scenarios in a struc-

tured and unified manner. DSLs improve productivity by raising the level of abstraction and shifting the user's focus to using domain abstractions [19]. Our work in this paper lays the groundwork for the future development of a DSL derived from the metamodel, enabling expressive and user-friendly scenario specifications.

3) **Forecasting Tools** enables engineers and data scientists to generate forecasts using statistical and machine learning methods within the validated specification boundaries. In addition, they support stakeholders in retrieving and managing the adjustments lineage.
4) **Analysis Dashboards** facilitate different stakeholders to visualize and interpret results, performance metrics, and check constraints compliance for different scenarios.

V. Experiment Design and Results

In this section, we first describe our experiment design and then discuss the three research questions that our evaluation addresses. For transparency and reproducibility purposes, our code and experiment data is available in our repository [20].

A. Experiment Design

Our proposed framework aims to support validation that extends beyond simple schema-level structural checks and facilitate the traceability of adjustments in scenario-based forecasting systems. Godahewa et al. introduce Monash Time Series Forecasting repository containing 30 datasets covering different domains, which includes both publicly available time series datasets and the datasets curated by them [21]. To the best of our knowledge, there is no open dataset available that directly models scenarios. Due to this limitation, we randomly sample nine real-world time-series datasets from the Monash repository (as shown in Table II) and mimic the baseline scenario to evaluate our framework. Moreover, the Monash repository was never seen or touched while performing the design activities of our proposed framework and metamodel. Godahewa et al. use the `.tsf` extension to store meta-information for their time-series datasets such as dataset name and frequency. Figure 4 presents our overall evaluation pipeline that addresses three research questions. Our pipeline is composed of four phases – *Data Collection and Preprocessing*, *Error Injection*, *Error Detection*, and *Traceability Construction*.

B. Research Questions

For each research question, we present our motivation, methodology, and results.

- **RQ1 (Metamodel Mapping Coverage)** To what extent can the classes and attributes of our proposed metamodel be mapped onto real-world datasets?

Motivation: Our proposed metamodel is central to our framework, ForeSPECT. This metamodel provides a formal structure that is essential for specification, validation, and traceability. This metamodel includes classes that capture domain concepts and their relationships within scenario-based forecasting systems. Therefore, it is important to assess how many of these classes can be mapped to concepts found in real-world forecasting systems.

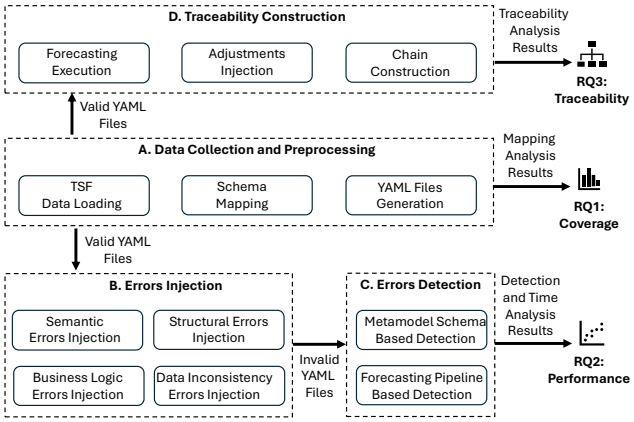


Fig. 4: Our Evaluation Pipeline

Methodology: Real-world forecasting systems are inherently complex and involve diverse data entities such as time-series records, pipeline specifications, performance metrics, business constraints, and decision parameters. Since our case studies are grounded in time-series datasets, this paper narrows its evaluation to the data characteristics related to time-series records. In our metamodel, *TimeSeriesProfile* and *Variable* classes primarily represent the data characteristics related to time-series datasets. Therefore, we use these classes to access the coverage of our metamodel. In addition, we consider *Scenario* and *ForecastingModel* classes because of the cross-entity references. As shown in Figure 4, we first use *TSF Data Loader* in the *Data Collection and Preprocessing* phase to fetch nine datasets from the Monash Time Series Forecasting repository using the Zenodo API and transform each dataset with the .tsf extension into a data frame. In addition, we extract meta-data such as series-type (univariate or multivariate) and source of these datasets. Next, we analyze each dataset and construct a mapping between the dataset metadata and the attributes of the classes that we study in our metamodel. Since all datasets share the same metadata structure, we automate the mapping step with a script in the *Schema Mapping* step. Finally, we generate YAML files and the mapping results in the *YAML Files Generation* step.

Table I: METAMODEL COVERAGE ANALYSIS

Category	Mapped Count	Total Count	Coverage (%)
Variable	8	9	88.9
TimeSeriesProfile	7	8	87.5
ForecastingModel	4	5	80.0
Scenario	6	11	54.5
Total Average	**6.25**	**8.25**	**77.72**

Results:

As shown in Table I, the *Scenario* and *ForecastingModel* classes have the maximum (11) and minimum (7) required attributes, respectively. Using a test dataset unseen during

design, our metamodel achieves 77.7% mapping coverage for classes describing time-series characteristics, including those with cross-entity references (e.g., *Scenario* referencing *Variable*). We dynamically map certain attributes using heuristics. For example, if *series-type* in *TimeSeriesProfile* is `UNIVARIATE`, then *method* in *ForecastingModel* must be a univariate model. We assign random static values to unmapped attributes such as *confidenceLevel* and *status* in *Scenario* to complete the remaining workflow.

Table II: ERROR DETECTION ACROSS DATASETS

Dataset	Domain	Series Type	Freq.	Errors
M1	Mixed	Univariate	Yearly	13
M3	Mixed	Univariate	Yearly	12
Tourism	Tourism	Univariate	Yearly	13
Solar	Energy	Multivariate	Weekly	12
SF Traffic	Transport	Multivariate	Hourly	12
Electricity	Energy	Multivariate	Weekly	12
Weather	Nature	Univariate	Daily	12
COVID	Nature	Multivariate	Daily	13
US Births	Nature	Univariate	Daily	13
Total Errors				**112**
Detected by Proposed Approach				**112**
Detected by Pipeline				**13**

- **RQ2 (Validation Efficiency)** How efficient is our model-driven approach in detecting errors early compared to relying on failures within the forecasting pipeline?

Motivation: In real-world forecasting workflows, YAML-based specification files often drive processes such as data ingestion, scenario setup, and model execution. Since these workflows consume computational resources, errors in these files can cause significant delays when the pipline fails eventually. In the worst-case scenario, these forecasting workflows may generate misleading results when the pipeline completes successfully without detecting the errors. Although our evaluation pipeline injects errors synthetically, it provides an approximation of the capability of our model-driven approach to capture errors early. This setup helps us understand whether semantic validation upfront improves both error detection rates and operational efficiency in practice.

Methodology: The valid YAML files generated from the *Data Collection and Preprocessing* phase are used in the *Errors Injection* phase. In this phase, we introduce synthetic errors that violate semantic, structural, business, and data consistency rules, as explained in Section IV-C. For example, setting *endDate* to an earlier time than the *startDate* violates the semantic rule which ensures that the end date cannot be earlier than the start date. We inject semantic, structural, business, and data inconsistency errors and create invalid YAML files for each error type. These files are first processed with our model-driven validation layer and then through a simple forecasting pipeline. This forecasting pipeline includes only the steps essential for processing time-series data such as data scaling before fitting the model. We use an automated timer that starts at the beginning of file processing stage and stops when either errors are detected using our approach (or pipeline-based method without our approach fails). If the stage completes without error detection, we do not record time. As the evaluation relies on synthetic rather than real errors, this poses a potential threat to the construct validity. We mitigate it by designing errors to mirror frequent forecasting workflow errors and validate their representativeness with domain knowledge.

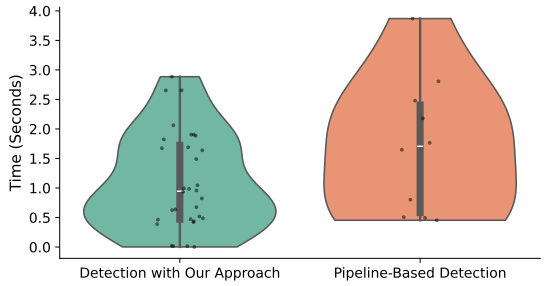


Fig. 5: Time Comparison for Error Detection

Results: As shown in Table II, our evaluation pipeline injects 112 errors. Error injection for some errors related to business rules fail due to missing attributes in the datasets. Our proposed approach successfully detects all injected errors, whereas the pipeline-based method without our approach detects only 13 errors. As illustrated in Figure 5, the mean time value for error detection with our approach ($\mu \approx 1.20$) is considerably lower than that of pipeline-based detection ($\mu \approx 1.90$). This indicates that our approach detects errors earlier on average, with a mean improvement of about 0.7 seconds per error. Moreover, we observe some extreme values in the pipeline-based detection method, such as 8811 and 450 seconds, when applied to one of the larger datasets. In contrast, the maximum time taken by our approach is 25 seconds, observed only once, likely due to delays in the data loading and processing stage. For a fair performance comparison, these outliers have been removed from both approaches.

- **RQ3 (Traceability)** Can forecasting adjustments be traced backward and forward using our approach?

Motivation: After generating forecast results, stakeholders such as analysts and domain experts often apply manual or automated adjustments to adapt the forecasts to business needs (e.g., scaling values and replacing outliers). Since these adjustments may alter the original forecast values (baseline) significantly, it becomes non-trivial to establish adjustments provenance. Without this provenance, it becomes difficult to track changes in a forecast over time. Based on our metamodel, each adjustment is composed of a traceability node that extends the *Traceable* interface, enabling chain-based reasoning over applied changes. Evaluating whether adjustments can be traced backward (from adjusted value to baseline) and forward (from baseline to adjusted value through successive adjustments) is essential to ensure traceability.

Methodology: In the *Traceability Construction* phase, we use the valid YAML files generated from the *Data Collection*

and Processing phase. These valid YAML specifications are first processed in a simple forecasting pipeline to generate forecasting results. Next, we randomly sample variables, time-series records, adjustment types, and adjustment factor for each adjustment. We encode these adjustments in the metamodel using their types (*adjustmentType*) and establish provenance meta-data through *traceID*, *parentTraceID*, and *depthLevel*. For each forecast series, we apply sequences of multiplicative, additive, and replacement adjustments while recording dependency references in the *Adjustments Injection* step. Using the chain mechanism of *TraceabilityNode* in the *Chain Construction* step, we evaluate (i) whether backward trace queries reconstruct the original baseline from the adjusted results and (ii) whether forward trace queries reproduce the final adjusted outcomes starting from the baseline.

Results: We apply a total of 76 adjustments across all datasets (25 `ADDITIVE`, 44 `MULTIPLICATIVE`, and 7 `REPLACEMENT`). The evaluation of forward trace queries indicates that adjusted values are fully reproducible in every case, achieving 100% forward traceability. On the other side, backward traceability reaches 91%. The shortfall is attributable to the `REPLACEMENT` adjustment type, as the original values cannot be recovered when navigating in reverse (e.g., once a value is overwritten, its original state is lost). This limitation arises because the adjustment factor encodes sufficient information for forward propagation but does not support backward reconstruction for `REPLACEMENT` adjustments.

VI. Conclusion and Future Work

Scenario-based forecasting systems must be flexible and coherent to adapt to changing conditions over time. Meeting these requirements makes it challenging to support stakeholders in performing validations that go beyond basic schema checks and in enabling traceability of forecasting decisions. In this paper, we introduce ForeSPECT, a model-driven framework that aims to improve the capabilities of existing specification formats like YAML by embedding semantic domain knowledge. Our framework detects semantic inconsistencies that remain undetected in YAML-based specifications and establishes provenance tracking for improving interpretability in forecasting workflows. The core of this framework is the metamodel that provides a formal structure to represent, validate, and trace forecasting activities. We evaluate our approach using nine unseen real-world datasets where our metamodel achieves 77.7% mapping coverage for classes representing time-series data characteristics. Also, our approach detects all synthetic errors with an average improvement of 0.7 seconds per error and achieves 100% forward and 91% backward traceability, with limitations in replacement adjustments.

In the future, first, we plan to extend our mapping study to cover the remaining classes in the proposed metamodel, requiring extensive datasets that capture scenarios, adjustments, and forecasting results. Second, we plan to evaluate our approach on additional datasets, including those that explicitly model scenarios in forecasting systems. Third, we plan to enhance the traceability layer by implementing lineage management and impact analysis components to support audit trails and diverse adjustment types. Fourth, we plan to extend the metamodel to support automated adjustments using logical formulas. Finally, our work in this paper lay the ground work for future development of a domain-specific language derived from the metamodel to provide stakeholders with an expressive and user-friendly scenario specification platform.

Acknowledgment

This research benefits from the insights and support of NAV CANADA's Corporate Performance and Sustainability team.

References

[1] P. Goodwin, S. Gönül, D. Önkal, A. Kocabıyıkoğlu, and C. I. Göğüş, "Contrast effects in judgmental forecasting when assessing the implications of worst and best case scenarios," *J. Behav. Decis. Mak.*, vol. 32, no. 5, pp. 536–549, 2019.

[2] F. Petropoulos, D. Apiletti, V. Assimakopoulos, M. Z. Babai, D. K. Barrow, S. B. Taieb, C. Bergmeir, R. J. Bessa, J. Bijak, J. E. Boylan, and et al., "Forecasting: Theory and practice," *Int. J. Forecast.*, vol. 38, no. 3, pp. 705–871, 2022.

[3] M. Dean, "Scenario planning: A literature review," *A report of project*, no. 769276-2, 2019.

[4] I. Predoaia, D. Kolovos, A. Garcia-Dominguez, M. Lenk, W. Ebel, and J. Burkl, "Towards processing yaml documents with model management languages," in *Proc. ACM/IEEE 27th Int. Conf. Model Driven Engineering Languages and Systems*, 2024, pp. 970–979.

[5] Pydantic, https://docs.pydantic.dev/latest/, Last Accessed 2025.

[6] J. S. Armstrong, "Evaluating forecasting methods," in *Principles of Forecasting*, 2001, pp. 443–472.

[7] T. Gneiting, "Making and evaluating point forecasts," *J. Am. Stat. Assoc.*, vol. 106, no. 494, pp. 746–762, 2011.

[8] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *Int. J. Forecast.*, vol. 22, no. 4, pp. 679–688, 2006.

[9] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*. OTexts, 2018.

[10] H. Hewamalage, K. Ackermann, and C. Bergmeir, "Forecast evaluation for data scientists: common pitfalls and best practices," *Data Min. Knowl. Discov.*, vol. 37, no. 2, pp. 788–832, 2023.

[11] S. Meisenbacher, M. Turowski, K. Phipps, M. Rätz, D. Müller, V. Hagenmeyer, and R. Mikut, "Review of automated time series forecasting pipelines," *WIREs Data Min. Knowl. Discov.*, vol. 12, no. 6, p. e1475, 2022.

[12] S. Kansab, "Machine learning pipeline for software engineering: A systematic literature review," *arXiv preprint arXiv:2508.00045*, 2025.

[13] G. Zhang, "A data traceability method to improve data quality in a big data environment," in *Proc. IEEE 5th Int. Conf. Data Sci. in Cyberspace (DSC)*, 2020, pp. 290–294.

[14] S. Kanwal, F. Z. Khan, A. Lonie, and R. O. Sinnott, "Investigating reproducibility and tracking provenance–a genomic workflow case study," *BMC Bioinformatics*, vol. 18, no. 1, p. 337, 2017.

[15] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Gener. Comput. Syst.*, vol. 25, no. 5, pp. 528–540, 2009.

[16] J. Freire, D. Koop, E. Santos, and C. T. Silva, "Provenance for computational tasks: A survey," *Comput. Sci. Eng.*, vol. 10, no. 3, pp. 11–21, 2008.

[17] H. Miao, C. Wang, and S. S. Bhowmick, "Provenance for batch and streaming ml workflows," in *Proc. IEEE 37th Int. Conf. Data Eng. (ICDE)*, 2021, pp. 2589–2594.

[18] A. Spinuso, M. Hardt, G. Sipos, P. Gubian, and L. Trani, "Enhancing provenance for reproducible time-series analysis in large-scale scientific workflows," in *Proc. IEEE 15th Int. Conf. e-Science*, 2019, pp. 246–255.

[19] S. Kelly and J.-P. Tolvanen, "Visual domain-specific modelling: Benefits and experiences of using metacase tools," in *Int. Workshop on Model Engineering, at ECOOP*, 2000, pp. 1–9.

[20] R. Saini, "Forespect," 2025. [Online]. Available: https://doi.org/10.5281/zenodo.17274587

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