

Article

BiTE: A Bitemporal Event-Centered Database Framework for Dynamic Aeronautical Information State Management

Tianyue Wei ¹, Xin Lai ^{1,*}, Yidan Liang ¹, Chengwei Zhang ² and Rui Kang ¹

¹ College of Air Traffic Management, Civil Aviation Flight University of China, Chengdu 641419, China; weitianyue@cafuc.edu.cn (T.W.); liangyidan@cafuc.edu.cn (Y.L.); kangrui@cafuc.edu.cn (R.K.)

² International Cooperation Department, Civil Aviation Flight University of China, Guanghan 618307, China; chengwei.zhang@cafuc.edu.cn

* Correspondence: laixin@cafuc.edu.cn

Abstract

Dynamic aeronautical information is still widely disseminated through textual Notice to Air Missions (NOTAMs), while message-oriented storage cannot directly maintain the evolving states of affected objects. The objective of this study is to determine whether NOTAM-derived object events can be organized into traceable bitemporal states that support accurate and efficient current and historical access. To this end, this paper proposes BiTE, a bitemporal event-centered database framework that connects object-level event evidence, historical state versions, and materialized current-state projections. By integrating business and system time with NOTAM-specific lifecycle rules, BiTE supports state maintenance, historical reconstruction, and source traceability. A MongoDB-based prototype was evaluated using 44,591 NOTAMs from five major U.S. aerodromes. Independent manual validation showed 96.14–100% agreement across object identification and lifecycle-maintenance tasks. Across 1500 manually verified queries, BiTE achieved F1 scores of 99.43% and 98.66% for current-state and airport-overview retrieval, respectively, and a historical hit rate of 95.80%, outperforming representative message-oriented, relational-bitemporal, and RDF-based implementations. Mean query latency remained below 3.7 ms, while functionally equivalent ablations confirmed the performance contribution of the layered architecture. These results demonstrate that BiTE enables accurate, traceable, and efficient object-state management for dynamic aeronautical information.

Keywords: aeronautical information management; dynamic aeronautical information; Notice to Air Missions; bitemporal database; event-centered modeling; natural-language query



Academic Editor: Sven Groppe

Received: 24 June 2026

Revised: 16 July 2026

Accepted: 20 July 2026

Published: 22 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Aeronautical information supports flight planning, air traffic management, airport operations, and aviation safety. It includes relatively stable information published through products such as the Aeronautical Information Publication (AIP), as well as dynamic information describing temporary or frequently changing operational conditions [1,2]. The Notice to Air Missions (NOTAM) is a primary carrier of such dynamic information and reports time-sensitive changes involving aeronautical facilities, services, procedures, hazards, and other operational elements [1–3].

NOTAMs are distributed in textual and increasingly structured digital forms. Digital NOTAM and the Aeronautical Information Exchange Model (AIXM) provide machine-interpretable representations that support information exchange, validation, database updates, and downstream processing [4–7]. Nevertheless, textual NOTAMs remain prevalent

in operational information workflows. Although partly standardized, their interpretation still depends on abbreviations, contextual information, and domain knowledge [2,3,8–10].

Conventional NOTAM databases generally preserve each notice as a message-level record containing identifiers, locations, validity times, and free text. This organization supports archiving and filtering but does not directly represent the affected aeronautical object, its evolving operational state, or the evidence responsible for that state [6,11,12]. Operational queries, however, are often state-oriented: users need to determine the current condition of an object, reconstruct its condition at an earlier time, and trace that condition to its causative notices.

Lifecycle relations further complicate state maintenance. NOTAMN, NOTAMR, and NOTAMC records respectively introduce, replace, and cancel operational conditions [3]. If these relations are treated only as links among messages, replaced or cancelled information may remain visible, while historical state transitions are difficult to reconstruct. Moreover, the operational validity of a condition may differ from the time at which it is recorded or superseded in the database. Effective state management must therefore distinguish business time from system time while preserving the original evidence [13–15].

Existing technologies address only parts of this problem. AIXM and Digital NOTAM define structured representations and exchange mechanisms, while bitemporal databases, event sourcing, and versioning support validity, change, and history. However, they do not specify how source notices, one-to-many object events, lifecycle transitions, historical versions, and current projections should be coordinated in a unified database. The novelty of this study therefore lies in integrating these mechanisms with NOTAM-specific object and lifecycle semantics. Its objective is to develop and evaluate a lifecycle-aware bitemporal framework for accurate, traceable, and efficient object-state access.

Accordingly, this study proposes BiTE, a Bitemporal Event-Centered Database Framework for Dynamic Aeronautical Information State Management. BiTE organizes normalized NOTAM-derived object events into bitemporal state histories and current-state projections while preserving links to the original notices. NOTAM-specific rules maintain these states under new, replacement, and cancellation operations, supporting current-state access, historical reconstruction, and source traceability. Event normalization is treated as an upstream input-adaptation process; the primary focus is persistent database organization and lifecycle-aware state maintenance.

BiTE complements rather than replaces AIXM and Digital NOTAM [5,16]. The framework is evaluated using a full operational NOTAM corpus, independent manual references, representative message-oriented, relational-bitemporal, and RDF-based implementations, and functionally equivalent ablations examining the storage–latency contributions of its materialized layers.

The main contributions of this study are as follows:

1. Textual NOTAM management is formulated as an object-level state-management problem in which one notice may generate multiple object events and state transitions.
2. A layered bitemporal event-centered framework is proposed to coordinate source-linked event evidence, ordered historical state versions, and materialized current-state projections.
3. A lifecycle-aware maintenance method is developed for NOTAMN, NOTAMR, and NOTAMC operations, supporting object-level replacement, cancellation, historical reconstruction, and source traceability.
4. The framework is evaluated through full-corpus implementation checks, independent manual validation, comparisons with representative alternative database organizations, and functionally equivalent ablation experiments.

2. Related Work

2.1. Dynamic Aeronautical Information Processing

Dynamic aeronautical information processing converts specialized aviation texts into structured information for operational use. Named entity recognition, domain knowledge resources, ontologies, and terminology systems form important foundations for aeronautical information intelligence [8]. Domain-specific natural language processing methods have also been applied to aviation safety reports and Chinese aviation incident texts for classification, entity recognition, and interactive analysis [17,18].

Research on textual NOTAMs covers error correction, classification, information extraction, and semantic parsing. A MacBERT-based method uses phonetic and morphological knowledge graphs to correct errors in Chinese NOTAMs [19]. Attention-based models have been applied to abbreviated NOTAM classification [20]. Self-supervised language models have been used to extract operational information from NOTAM texts [9]. Knowledge-graph-based methods combine entity recognition, classification, and structured retrieval to support automated Chinese NOTAM processing [12]. The Knots dataset and its large language model prompting framework extend this line of research to expert-annotated NOTAM semantic parsing [10].

Semantic representation is another important area of research. Ontology-based models support the representation and semantic querying of Digital NOTAMs [6]. AIRM-based methods enable fine-grained semantic filtering of NOTAM information [7]. Shared semantic models have been used to integrate heterogeneous air traffic management data [21]. A spatio-temporal RDF Schema organizes NOTAM information through time, event, and location elements [11]. Structured NOTAM information has also been used in operational management and automated visualization [22,23].

Most existing studies focus on interpreting individual messages or converting them into semantic representations. Less attention has been given to organizing the extracted changes as persistent and evolving operational states. Dynamic aeronautical information therefore remains distributed across separate records, which limits current-state access, historical reconstruction, and source traceability.

2.2. Temporal and Bitemporal Data Management

Temporal data management extends conventional databases by recording how facts change over time. Valid time describes when a fact holds in the modeled domain. Transaction time records when the fact is stored or maintained in the database [24,25]. Conceptual temporal models define these two dimensions independently of their physical database implementation [14,15]. SQL:2011 introduced application-time periods and system-versioned tables, allowing facts to be queried by their operational validity or recorded history [26].

Efficient temporal access depends on storage structures and indexing methods. Partial-persistence and double-tree methods were developed for bitemporal time-slice queries [27]. Comparative studies examined access methods for time-evolving data [28]. Later work evaluated the performance of bitemporal database systems [29] and introduced timeline indexes for temporal selection and aggregation [30]. Transaction-time support has also been integrated into database engines [31]. Specialized representations have been proposed for open-ended intervals and the temporal value “now” [32].

Temporal semantics have also been introduced into RDF and knowledge-based systems. Time can be attached to graphs, statements, or resources [33]. Temporal RDF models support time-qualified facts and historical queries [34,35]. Bitemporal RDF combines valid time and transaction time to reconstruct domain states and previously recorded knowledge [36,37].

Most temporal and bitemporal models are designed for general data management. They provide limited support for domain-specific lifecycle rules, continuous object-state maintenance, and traceability across related operational records.

2.3. Event-Based State Evolution and Version Management

Event-based data management records operational changes as ordered events and derives system states from the event history. Persistent event logs preserve the sequence of changes and support event replay, failure recovery, and state reconstruction [38]. This structure separates change evidence from materialized state views and allows current states to be rebuilt without removing earlier records.

Temporal database research has also examined the relation between events and state evolution. Bitemporal event calculus combines event occurrence with valid time and transaction time. It supports state inference while preserving both operational and recorded histories [39]. Version-based methods retain successive database states instead of overwriting earlier values, which supports historical access and temporal consistency [40].

Event-based and bitemporal methods have been applied to business processes and connected data structures. Bitemporal process models preserve changing operational conditions and support process-state analysis under different historical contexts [41,42]. Bitemporal property graphs represent changes in entities and relationships [43]. Integrated event-state models show how change events and persistent state representations can be maintained under two temporal dimensions [44].

These studies provide a basis for event preservation, state reconstruction, temporal versioning, and historical querying. Their models are mostly domain-independent and offer limited guidance for converting operational messages into object-level state transitions. Replacement and cancellation rules, state closure, current-state projection, and evidence traceability are rarely handled within a unified data management process.

3. Database Model

3.1. Problem Setting

Dynamic aeronautical information describes operational changes affecting identifiable aeronautical objects, including runways, taxiways, navigation aids, procedures, facility services, and airspace activities. This study treats each affected aeronautical object as the basic unit of state management and represents its operational changes through temporally valid and traceable state records.

In this study, dynamic aeronautical information state management refers to the organization and lifecycle-aware maintenance of normalized object-level events derived from source notices as bitemporal state records with explicit traceability [45]. A source notice provides the original evidence; a business event represents a normalized object-level state change; a state version records an object state over business-time and system-time intervals; and a current projection provides direct access to the latest version eligible at a specified bitemporal query point. The core concepts are summarized in Table 1.

Table 1. Core concepts for database modeling.

Concept	Definition	Role in Database Modeling
Source Notice	Original dynamic aeronautical message	Provides raw evidence
Aeronautical Object	Operational entity affected by a notice	Serves as the unit of state management
Business Event	Structured state-change record	Drives object-state updates
State Version	Object-state snapshot within a validity interval	Supports historical reconstruction
Current Projection	Object state at a reference time	Supports operational state queries
Traceability Chain	Links notice, event, state version, and current state	Supports explanation and audit

Multiple notices may create, replace, or cancel states associated with the same aeronautical object. The proposed model therefore separates source evidence, state-change events, historical state versions, and current projections while preserving explicit links among these components.

3.2. Event-State Model

The proposed database model follows an event-state view. A source notice is not treated as the state of an aeronautical object. It is first transformed into a business event that describes the corresponding state change. The event links the source notice with the affected object, structured state content, and temporal information required for database maintenance [44].

A business event is represented as follows:

$$e = \langle i, a, n, o, p, T_b, T_s \rangle, \quad (1)$$

where i is the event identifier, a is the event action, n is the source notice, o is the affected aeronautical object, and p is the structured payload. The payload stores normalized state information such as status, reason, affected segment, facility type, and procedure amendment. T_b represents business validity, while T_s represents system validity.

For a state version s_i , the two temporal intervals are defined as follows:

$$T_b(s_i) = [t_b^{start}, t_b^{end}], \quad (2)$$

$$T_s(s_i) = [t_s^{start}, t_s^{end}]. \quad (3)$$

Here, $T_b(s_i)$ defines the period during which the state is effective in the operational environment. t_b^{start} and t_b^{end} are its business start and end times. $T_s(s_i)$ defines the period during which the state version is valid in the database. t_s^{start} and t_s^{end} are its system start and end times.

Business time records when a condition is operationally effective, while system time records when the corresponding state version is maintained in the database. The two periods may differ because notices can be received or processed after their effective time. Replacement and cancellation notices may close earlier versions without removing their histories, so a single temporal dimension is insufficient.

State maintenance also requires a stable object key that links notices affecting the same aeronautical object. The key must use suitable granularity to avoid merging distinct objects or splitting one object into separate histories.

BiTE maintains state changes through version creation and closure rather than direct overwriting [40]. An event may create a version, close an existing version, or perform both operations. The current projection is selected from the version history under the required business-time and system-time conditions [46]. The event-state process is summarized as follows:

Source Notice → Business Event → State Version → Current Projection

3.3. Design Requirements

The event-state model defines five requirements for the database design.

- R1.** Object-level state projection. Source notices should be transformed into object-level state records. This allows direct queries of the current and historical states of runways, taxiways, navigation aids, procedures, facility services, and airspace activities [47].

- R2.** Bitemporal validity management. The database should preserve both business validity and system validity. Business validity describes when a state is operationally effective, while system validity records when the state is maintained in the database.
- R3.** Event-driven versioning. State changes should be maintained through business events instead of direct overwriting. New, replacement, and cancellation notices create or close state versions without removing historical records.
- R4.** Source traceability. Each current or historical state should retain links to its business event and source notice. These links support explanation, auditing, and validation [48].
- R5.** Type-aware extensibility. The database should support different payload structures for different aeronautical information types while using the same event-state maintenance process.

These requirements guide the framework presented in the next section. The framework models source notices, business events, state versions, and current projections as separate but connected components.

4. Bitemporal Event-Centered Database Framework

4.1. Overview

BiTE organizes dynamic aeronautical information through four connected components: source notices, event normalization, the bitemporal event-centered database, and state maintenance. Figure 1 presents the overall architecture. Source notices provide the original operational evidence. Event normalization converts heterogeneous notices into object-level business events. The database core stores event evidence and object-state versions. State maintenance updates current-state projections according to NOTAM lifecycle operations.

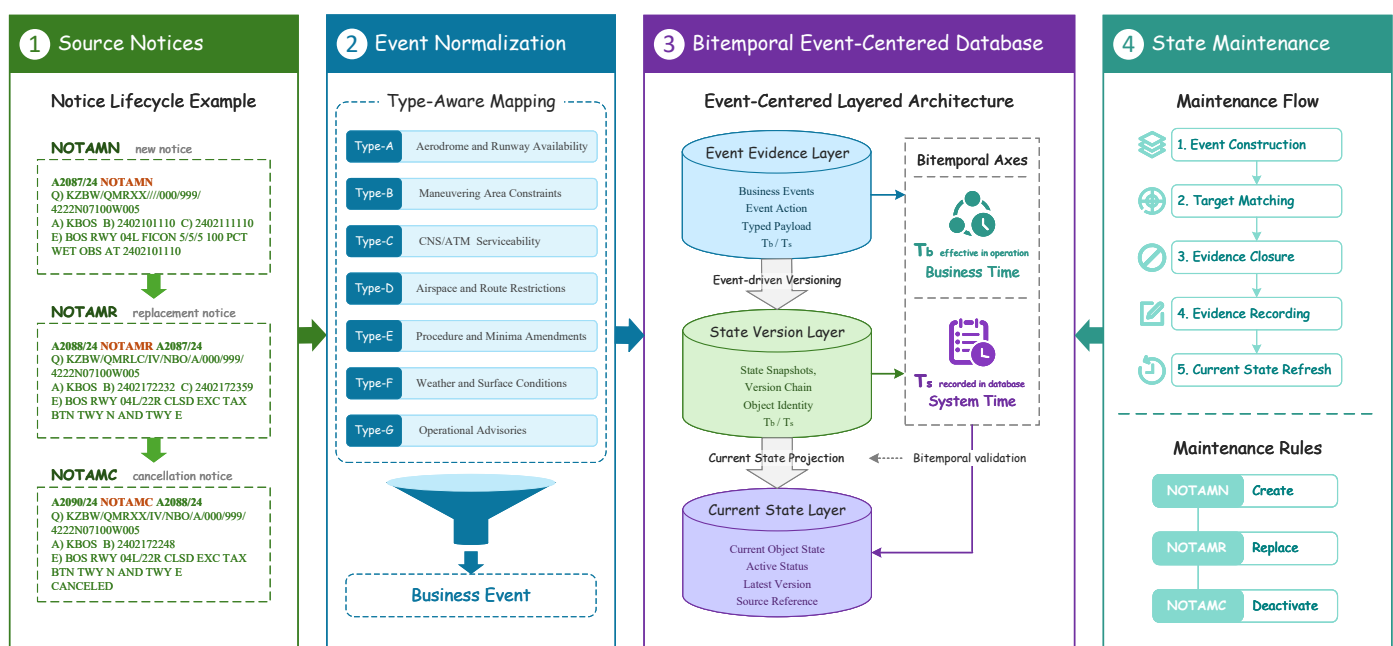


Figure 1. Overall architecture of the BiTE framework.

The source-notice module represents the lifecycle relations among new NOTAMs (NOTAMN), replacement NOTAMs (NOTAMR), and cancellation NOTAMs (NOTAMC). A NOTAMN introduces an operational condition, a NOTAMR replaces an earlier notice, and a NOTAMC cancels a previous notice. These relations allow the database to preserve the latest state and the evidence behind each state transition.

The event-normalization module converts each source notice into a type-aware business event. The event contains the lifecycle action, affected object, structured payload, and temporal validity. It is also assigned business time and system time before entering the database core [49]. Detailed type mapping and event construction are described in the next subsection.

The database core uses a three-layer structure. The Event Evidence Layer stores normalized business events together with source links, event actions, typed payloads, and temporal validity. The State Version Layer maintains state snapshots and version histories for affected objects. The Current State Layer stores directly accessible projections of the latest valid object states.

The bitemporal model separates operational validity from database validity. Business time describes when a condition is effective in the operational environment. System time records when the corresponding state version is valid in the database. This distinction preserves earlier evidence after replacement or cancellation and supports historical reconstruction without overwriting previous records.

The state-maintenance module processes each event according to its lifecycle action. It identifies the affected object, matches referenced targets, closes earlier evidence and state versions when required, creates new versions, and refreshes the current projection. A NOTAMN creates a state, a NOTAMR replaces an existing state, and a NOTAMC deactivates the referenced state.

4.2. Event Normalization

BiTE treats normalized object-event records as the input to database construction. Before entering the Event Evidence Layer, heterogeneous source notices are converted into a common event representation because different aeronautical objects and operational changes require different state fields. For example, runway closures, navigation-facility outages, airspace activities, and procedure amendments cannot share an identical payload schema. This normalization module provides structured inputs to BiTE but is not itself a core contribution of the database framework.

Let m denote a source notice and $\mathcal{C}$ the predefined set of information types

$$\mathcal{C} = \{\text{Type-A, Type-B, Type-C, Type-D, Type-E, Type-F, Type-G}\}. \quad (4)$$

Information-type mapping uses a learned classifier that jointly considers the Q-code and the free-text Item E. Let $q(m)$ denote the Q-code and $e(m)$ the Item E text. For each content-bearing notice, the notice-level routing type is defined as

$$\kappa(q(m), e(m)) = c, \quad c \in \mathcal{C}, \quad (5)$$

where $\kappa(\cdot)$ selects the schema family applicable to the notice. The Q-code provides a standardized prior for the affected subject and operational condition, whereas Item E contains the object identity, detailed status, and contextual evidence needed to distinguish specific operational scenarios. Location and validity-time fields remain important for event construction but are not used as the primary evidence for type classification.

At the implementation level, the Q-code and Item E are separately encoded using a pretrained RoBERTa-base encoder [50]. Their contextual representations are fused through a bidirectional cross-attention fusion (BiCAF) module, pooled, and passed to a linear softmax layer for seven-class prediction. This design uses the standardized Q-code as prior evidence to guide attention toward type-relevant text, while allowing the semantic evidence in Item E to compensate for generic, missing, or inaccurate codes. The resulting type determines the applicable payload schema: Type-A covers aerodrome and runway

availability; Type-B, maneuvering-area constraints; Type-C, CNS/ATM serviceability; Type-D, airspace and route restrictions; Type-E, procedure and minima amendments; Type-F, weather and surface conditions; and Type-G, operational advisories and facility services.

The notice-level type determines the applicable schema but does not imply a one-to-one relationship between notices and events. After type routing, notice m is decomposed into one or more object-level business events

$$\mathcal{B}(m) = \{b_{m,1}, b_{m,2}, \dots, b_{m,n_m}\}, \quad n_m \geq 1, \quad (6)$$

$$\text{type}(b_{m,j}) = \kappa(q(m), e(m)), \quad j = 1, \dots, n_m, \quad (7)$$

where each $b_{m,j}$ represents the effect of notice m on one aeronautical object or object segment. The derived events inherit the notice-level routing type but retain distinct canonical object keys and object-specific payloads. Consequently, one notice may generate several event rows when it affects multiple runways, taxiways, facilities, procedures, or object segments. For cancellation-only notices that introduce no new payload, the applicable type and affected objects are inherited from the resolved lifecycle target.

Header-derived fields, including the aerodrome, validity interval, operation type, and referenced notice, are parsed deterministically. The free-text payload is processed by a schema-constrained, type-aware generative language model conditioned on the selected type and Item E. It populates only the applicable fields, such as operational status, reason, affected segment, facility attributes, and procedure amendments. Deterministic post-processing then normalizes and validates object identifiers, status values, temporal fields, and lifecycle references. Accordingly, the Q-code serves as a routing prior rather than the sole source of normalized information, because it does not consistently represent the object granularity and type-specific attributes required for state maintenance.

Each normalized event contains the information type, lifecycle action, canonical object identity, typed payload, business-time interval, system-time interval, and source-notice reference. NOTAMN introduces a new object condition, NOTAMR replaces a referenced condition, and NOTAMC terminates a referenced condition. Business time represents when the condition is operationally effective, whereas system time represents when the corresponding event becomes visible in the database.

The normalized events are stored in the Event Evidence Layer and provide the basis for state-version construction, current-state projection, lifecycle processing, and source traceability. Detailed classifier and extraction-model development is outside the scope of this database study; the reliability of the resulting inputs is instead examined through the independent manual-validation results for information-type mapping and canonical object-key semantics reported in Section 6.2.

4.3. Database Architecture

The event-state model defines the basic modeling units. This section describes how these units are organized in BiTE. The database follows a layered architecture that separates event evidence, state versions, and current states. This design keeps source evidence stable and supports efficient state updates [37].

BiTE consists of the Event Evidence Layer, State Version Layer, and Current State Layer. The Event Evidence Layer records the events that cause state changes. The State Version Layer stores the historical states of affected objects. The Current State Layer provides direct access to the latest valid state. Table 2 summarizes the content and function of each layer.

The Event Evidence Layer stores normalized business events derived from source notices. Each event contains the source reference, lifecycle action, affected object, typed payload, and bitemporal validity. This layer preserves the evidence behind the creation,

replacement, or cancellation of an object state. Existing records are not overwritten when later notices arrive. A replacement or cancellation closes the validity of the referenced evidence and adds a new event record.

Table 2. Conceptual layers of the BiTE framework.

Layer	Main Content	Function
Event Evidence Layer	Business events, source notice references, lifecycle actions, typed payloads, bitemporal validity	Event evidence preservation and source traceability
State Version Layer	Object identities, state snapshots, version chains, validity intervals, event references	State versioning and historical reconstruction
Current State Layer	Current object state, active status, latest version reference, source-reference information	Efficient current-state access

The State Version Layer maintains an ordered version history for each affected object. A new notice creates a state version. A replacement notice closes the referenced version and appends its replacement. A cancellation notice closes the active version associated with the referenced condition.

For an affected object o , the ordered state history is

$$H(o) = \langle s_1, s_2, \dots, s_k \rangle, \quad (8)$$

where each state version s_i has a unique, monotonically increasing order $\text{ord}_o(s_i) \in \mathbb{N}$, such that $\text{ord}_o(s_i) < \text{ord}_o(s_j)$ for $i < j$. This order defines a strict total order over $H(o)$ and supports deterministic selection when validity intervals overlap. Each version stores the state snapshot, object identifier, source event, and bitemporal validity intervals [51].

The Current State Layer provides direct access to the latest eligible state of each aeronautical object. Let $\tau = (\tau_b, \tau_s) \in \mathbb{T}_b \times \mathbb{T}_s$ denote a bitemporal query point, where τ_b and τ_s are the business-time and system-time coordinates. For version s_i , let $T_b(s_i) = [b_i^-, b_i^+)$ and $T_s(s_i) = [\sigma_i^-, \sigma_i^+)$, with $+\infty$ representing an open-ended upper bound. The eligible versions are

$$\mathcal{E}(o, \tau) = \{s_i \in H(o) \mid \tau_b \in T_b(s_i) \wedge \tau_s \in T_s(s_i)\}. \quad (9)$$

The current-state projection is defined as

$$C(o, \tau) = \begin{cases} \arg \max_{s_i \in \mathcal{E}(o, \tau)} \text{ord}_o(s_i), & \mathcal{E}(o, \tau) \neq \emptyset, \\ \emptyset, & \mathcal{E}(o, \tau) = \emptyset. \end{cases} \quad (10)$$

Thus, if multiple versions satisfy both temporal conditions, the unique version with the highest object-level order is selected; if none is eligible, the object has no state at τ . The Current State Layer materializes this bitemporal projection while retaining links to event evidence and source notices. The same eligibility and selection rules are used when the projection is reconstructed from the State Version Layer.

4.4. State Maintenance

State maintenance applies each normalized event to the Event Evidence, State Version, and Current State layers. A NOTAMN creates a new object state, a NOTAMR closes the referenced state in system time and creates its replacement, and a NOTAMC closes the referenced state without creating a new version.

Let b_e^- and b_e^+ denote the business start and end times derived from Items B and C, and let σ_e denote the database processing time. Event e is assigned

$$T_b(e) = [b_e^-, b_e^+), \quad T_s(e) = [\sigma_e, +\infty). \quad (11)$$

For a matched record x , $\text{Close}(x; \sigma_e)$ preserves its business interval $T_b(x) = [b_x^-, b_x^+)$ and updates its system interval to $T_s(x) = [\sigma_x^-, \min(\sigma_x^+, \sigma_e))$. Thus, replacement or cancellation ends the record's system validity at the processing time without changing the business end declared in Item C. The closed record remains available for queries with an earlier system-time coordinate.

Algorithm 1 is applied independently to each object-level event $b_{m,j}$ generated from a source notice. It first binds the event to its object and bitemporal intervals and stores it in the Event Evidence Layer. A NOTAMN creates a state version inheriting these intervals and refreshes the current projection (lines 6–9). A NOTAMR closes the matched evidence and predecessor version at σ_e , creates a replacement with $T_b = [b_e^-, b_e^+)$ and $T_s = [\sigma_e, +\infty)$, and refreshes the projection (lines 10–19). A NOTAMC applies the same system-time closure and records the cancellation relation without creating a replacement version (lines 20–28). In all three cases, Refresh recomputes $C(o, \tau^{\text{cur}})$ from $H(o)$ using Equations (9) and (10); it does not unconditionally assign the newly appended version as the current state.

Algorithm 1 Bitemporal Event-Driven State Maintenance

Input: source notice m ; normalized event e ; event evidence layer E ; state version layer H ; current state layer C

Output: updated E , H , and C

```

1:  $op \leftarrow \text{Op}(m)$ 
2:  $o \leftarrow \text{Obj}(e)$ 
3:  $(b_e^-, b_e^+, \sigma_e) \leftarrow \text{Time}(m, e)$ 
4:  $e \leftarrow \text{Bind}(e, o, [b_e^-, b_e^+), [\sigma_e, +\infty))$ 
5:  $E \leftarrow \text{Add}(E, e)$ 
6: if  $op = \text{NOTAMN}$  then
7:    $s \leftarrow \text{NewState}(e)$ 
8:    $H(o) \leftarrow \text{Append}(H(o), s)$ 
9:    $C(o) \leftarrow \text{Refresh}(s)$ 
10: else if  $op = \text{NOTAMR}$  then
11:    $q_m \leftarrow \text{LookupReference}(E, \text{Ref}(m))$ 
12:    $(q_E, q_H) \leftarrow \text{AlignObjectTarget}(q_m, o, m)$ 
13:   if  $q_E \neq \emptyset \wedge q_H \neq \emptyset$  then
14:      $E \leftarrow \text{Close}(E, q_E, \sigma_e)$ 
15:      $H(o) \leftarrow \text{Close}(H(o), q_H, \sigma_e)$ 
16:      $s \leftarrow \text{Replace}(e, q_H)$ 
17:      $H(o) \leftarrow \text{Append}(H(o), s)$ 
18:      $C(o) \leftarrow \text{Refresh}(s)$ 
19:   end if
20: else if  $op = \text{NOTAMC}$  then
21:    $q_m \leftarrow \text{LookupReference}(E, \text{Ref}(m))$ 
22:    $(q_E, q_H) \leftarrow \text{AlignObjectTarget}(q_m, o, m)$ 
23:   if  $q_E \neq \emptyset \wedge q_H \neq \emptyset$  then
24:      $E \leftarrow \text{Close}(E, q_E, \sigma_e)$ 
25:      $H(o) \leftarrow \text{Close}(H(o), q_H, \sigma_e)$ 
26:      $E \leftarrow \text{Link}(E, e, q_E, \text{cancels})$ 
27:      $C(o) \leftarrow \text{Deactivate}(C(o), \sigma_e)$ 
28:   end if
29: end if
30: return  $E, H, C$ 

```

Target resolution proceeds in two stages. The explicit reference identifier in a NOTAMR or NOTAMC first locates the referenced source notice; it is the primary notice-level lookup key rather than an input to similarity-based matching. The derived events and state versions are then aligned at the object level because one referenced notice may produce multiple object-level events. Canonical object identity performs this alignment, while aerodrome and Q-code subject serve only as consistency constraints. If the referenced notice is unavailable or no corresponding object-level pair can be established, the new event remains in *E* without modifying *H* or *C*. Closed records remain available for bitemporal historical retrieval [52].

5. Experiments

5.1. Experimental Objectives

This section evaluates whether BiTE can construct and maintain object-level dynamic aeronautical information states and provide accurate and efficient access to current and historical information. The core experiments cover database construction, independently referenced semantic correctness, comparative object-state retrieval, and architectural contribution. Full-corpus implementation checks are distinguished from correctness measured against manually established references.

The evaluation addresses the following questions.

- Q1.** Database construction. Can BiTE organize normalized object-level events into state records while preserving the required temporal and traceability links?
- Q2.** State-maintenance correctness. Can BiTE correctly identify affected objects and maintain their evolving states under different NOTAM lifecycle operations?
- Q3.** Object-state retrieval. Can BiTE provide accurate and efficient access to current, airport-overview, and historical object states compared with representative data-management approaches?
- Q4.** Architectural contribution. How do the Current State Layer, State Version Layer, and lifecycle-maintenance mechanism affect query performance, storage cost, historical reconstruction, and the exclusion of obsolete states?

A separate auxiliary experiment additionally examines the feasibility of converting informal natural-language questions into structured query intents for the BiTE query interface [53]. It is not treated as part of the core validation of database correctness or architectural effectiveness.

5.2. Dataset and Implementation

5.2.1. Evaluation Corpus

The evaluation corpus was obtained from a historical operational repository following International Civil Aviation Organization (ICAO) NOTAM formats. It contains all NOTAMs issued in 2024 for five major U.S. aerodromes: Hartsfield–Jackson Atlanta International Airport (KATL), Boston Logan International Airport (KBOS), Charlotte Douglas International Airport (KCLT), John F. Kennedy International Airport (KJFK), and Los Angeles International Airport (KLAX). These aerodromes generated substantial notice volumes and contained sufficient new, replacement, and cancellation relations for evaluating object-state construction and lifecycle maintenance. The full corpus comprises 44,591 source NOTAMs.

5.2.2. Implementation and Statistics

BiTE was implemented on MongoDB. The normalized events and state versions share common fields for object identity, lifecycle action, temporal validity, and source references, while their operational payloads vary across information types. MongoDB was selected because its document model accommodates heterogeneous nested payloads and supports

compound indexing over the principal query fields without requiring extensive sparse tables or frequent schema modifications.

The selection of MongoDB is an engineering choice rather than an inherent constraint of the BiTE conceptual framework. Compared with key–value and wide-column stores, its document structure and indexing mechanisms provide more direct support for the multi-attribute temporal queries and nested payloads required in this study. Compared with graph databases, it allows materialized current-state projections to be retrieved without repeatedly traversing event–state relationships. Nevertheless, the Event Evidence, State Version, and Current State layers could also be implemented using relational or graph-based databases. The comparative experiments therefore evaluate the suitability of the MongoDB-based implementation for the studied workload rather than claiming its universal superiority over other database technologies.

The implementation stores source notices, normalized notices, event evidence, state versions, and current-state projections in separate collections. Each state version retains references to its causative business event and source notice, while the Current State collection materializes the latest eligible object-state projection. Table 3 summarizes the database constructed from the full corpus.

Table 3. Airport-level dataset and database statistics.

Airport	Source NOTAMs	Object-Level Events	Objects	State Versions	Current States
KBOS	12,780	15,879	188	8857	117
KATL	8752	12,261	336	7353	204
KJFK	8291	9101	259	5003	134
KCLT	7644	11,092	378	6006	140
KLAX	7124	7362	254	4362	156
Total	44,591	55,695	1415	31,581	751

The number of object-level events exceeds the number of source NOTAMs because one notice may affect multiple aeronautical objects. The number of state versions is lower because cancellation, closure, and other non-state-producing events do not create new state snapshots. After lifecycle processing, 751 object states remained active at the evaluation time.

5.2.3. Manual Annotation and Reference Sets

A fixed random seed of 2026 was used for all sampling. To establish an external reference independent of the BiTE-generated records, 1000 source NOTAMs were sampled from the full corpus, with 200 notices selected from each aerodrome. Manual examination and object-level decomposition produced 1971 object-event reference instances. Of these, 1739 were state-producing instances for which a state version was expected, while the remaining 232 represented cancellation- or closure-only operations that modified existing states without creating new versions.

The annotation was independently conducted and cross-checked by three experienced airline dispatchers. For each sampled case, the dispatchers identified the information type, affected object, canonical object key, lifecycle target, expected state-version operation, version-closure decision, and causative source NOTAM. Inter-annotator agreement before consensus adjudication was assessed using Fleiss' κ [54], yielding $\kappa = 0.829$, which indicates strong agreement among the three annotators [55]. Disagreements were subsequently resolved through joint re-examination of the source notice and its lifecycle context. The final reference labels were therefore determined from the source evidence rather than copied from the constructed database.

Three additional reference sets were prepared for query-level evaluation. Each contained 500 manually verified cases, yielding 1500 queries in total: 500 current object-state queries, 500 airport current-overview queries, and 500 historical-state queries. The Gold results specified the valid state of the target object, the active state set of the selected aerodrome, or the applicable historical state under the given temporal conditions. The same reference sets were used to evaluate BiTE and all comparison methods.

5.2.4. Runtime Environment

The experiments were conducted on Windows 11 AMD64 with 32 logical processors. The software environment comprised Python 3.12.3, PyMongo 4.17.0, pandas 2.3.2, and MongoDB 8.0.5 with the WiredTiger storage engine. The main indexes covered object keys, aerodromes, object types, activity flags, version numbers, and bitemporal validity fields. Representative MongoDB explain plans confirmed indexed execution without collection scans.

5.3. Experimental Design

5.3.1. Database Construction and Independent Validation

The database-construction experiment processed the full corpus through the BiTE pipeline to generate object-level business events, state versions, and current-state projections. Object-key assignment completeness, state-version materialization completeness, and traceability-link completeness were evaluated over the full corpus as implementation integrity checks. These indicators measured whether the required records and links were generated but were not treated as independent evidence of semantic correctness.

Semantic correctness was evaluated separately against the manually established reference set. The evaluation examined information-type assignment, canonical object-key construction, lifecycle-target resolution, state-version creation, version-closure decisions and source attribution. The reference labels were determined from the source NOTAMs and their lifecycle context rather than from the records generated by BiTE.

5.3.2. Baseline Implementations and Query Workload

Four database implementations were compared. Message-Mongo stored source notices as message-oriented documents and reconstructed object states during query execution. SQL-Bitemporal used an SQL:2011-style relational representation of objects, state records, and bitemporal intervals. RDF-NOTAM represented notices, objects, lifecycle relations, temporal conditions, and provenance as RDF statements. Full BiTE used the proposed layered object-state organization. All methods shared the same corpus, query conditions, runtime environment, and manually established reference sets, and their outputs were normalized to a common object-level representation. These implementations serve as representative architectural baselines rather than exact reproductions of individual published systems.

The workload covered three recurring aviation-information access patterns: point retrieval of an object's current state, airport-level retrieval of all active states, and bitemporal reconstruction of a historical state. Each task contained 500 manually verified cases. Current and overview results were compared with complete reference sets, while historical retrieval was successful when the returned set contained the manually identified version. The workload therefore targets BiTE's principal state-access functions rather than general analytical or concurrent database processing.

Under the warm-cache protocol, each query was executed once for warm-up and then measured over five rounds, producing 2500 latency observations for each task–method pair. End-to-end latency included database execution, cursor retrieval, result reading, and method-specific state reconstruction.

5.3.3. Functionally Equivalent Ablation Design

The layer-ablation experiment was redesigned to preserve equivalent state semantics between Full BiTE and the ablated variants. Removing a materialized layer altered only the retrieval path, while lifecycle rules, bitemporal eligibility, version selection, and normalized output semantics remained unchanged.

For current object-state and airport current-overview queries, Full BiTE accessed the Current State Layer, whereas the variant without this layer reconstructed the same projections exclusively from the State Version Layer under identical temporal conditions. For historical queries, Full BiTE accessed the State Version Layer, while the corresponding ablation reconstructed historical states by replaying Event Evidence records under the same lifecycle and temporal rules.

Before latency comparison, all outputs were normalized and compared by object identity, operational status, state payload, temporal validity, and source reference. Latency and storage were evaluated only after semantic equivalence was confirmed, with storage measured using MongoDB collection statistics. A separate lifecycle-maintenance ablation intentionally disabled replacement and cancellation effects to quantify the return of obsolete states; it was therefore not treated as functionally equivalent to Full BiTE.

The same warm-cache protocol was used as in the comparative evaluation. Each of the 500 queries was executed once for warm-up and five times for measurement, yielding 2500 observations per task–method pair. Execution order was randomized, and end-to-end latency included database retrieval and reconstruction.

5.3.4. Auxiliary Natural-Language Query-Intent Experiment

As an auxiliary experiment, a separate set of 150 bilingual questions was constructed with reference to recurring information requests from airline dispatchers and other frontline aviation operations personnel. The set was evenly divided among current object-state, airport current-overview, and historical-state queries. Each task contained 25 Chinese and 25 English questions. To approximate operational communication, the questions used relatively informal wording, implicit conditions, different levels of specificity, and varied expressions of object status and time.

Each question was manually assigned a Gold intent specifying the query scope, aerodrome, object type, target reference, status condition, business time, and system time. The Gold intents were established and checked against the intended database access conditions before model evaluation. All parsers received the same questions and produced intents following the same structured schema.

Seven representative contemporary LLMs from major model families were evaluated as optional query-intent parsers: DeepSeek-V4-Flash, Gemma 3 Instruct [56], Llama 3.2 Instruct, Phi-4 Mini Instruct [57], Qwen3-VL Instruct [58], Qwen3.5 Instruct, and GPT-5.4 nano. A deterministic rule-based parser was included as a simple baseline and used bilingual dictionaries, keywords, and regular expressions to identify aerodromes, query scopes, object types, target references, status conditions, and explicit temporal expressions. This experiment evaluates the feasibility of structured intent generation as an optional access mechanism rather than database-state correctness or usability improvements over conventional query interfaces. Exact model identifiers, providers, decoding settings, and the common prompt structure are provided in Appendix A.

5.4. Evaluation Metrics

The evaluation covers full-corpus implementation integrity, correctness against independent manual references, object-state query effectiveness and efficiency, architectural ablation, and auxiliary query-intent parsing. Structural integrity indicators derived from

the constructed database are distinguished from semantic correctness measured against manually established Gold references.

5.4.1. Implementation Integrity and Independent Correctness

Full-corpus implementation integrity is described using the key-assignment rate, versioning rate, traceability-link rate, and active projection ratio. These indicators respectively report the proportions of eligible construction cases receiving non-empty object keys, expected state records being materialized, state versions retaining the required event and source links, and identified objects remaining active at the evaluation time. They measure internal structural completeness rather than independent semantic correctness; the active projection ratio is treated as a descriptive database statistic.

Independent semantic correctness is evaluated against the manually established references. Information-type mapping, canonical object-key construction, lifecycle resolution, version closure, and source attribution use the general accuracy measure

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^N I(\hat{y}_i = y_i^*), \quad (12)$$

where $\hat{y}_i$ is the output produced by BiTE, y_i^* is the corresponding Gold label, and $I(\cdot)$ is the indicator function. Depending on the evaluated aspect, the label represents an information type, canonical object key, lifecycle relation, version-closure decision, or causative source NOTAM. An object key is considered correct only when its aerodrome, object type, normalized identifier, and applicable granularity components match the Gold key. State-version creation recall is the proportion of manually expected state versions successfully generated and matched by BiTE.

The evaluation units follow the level at which each decision is made: type mapping is evaluated at the source-notice level, object-key semantics at the object-event level, lifecycle resolution at the lifecycle-decision level, and version creation, closure, and source attribution at the state-version level.

5.4.2. Object-State Query Effectiveness and Efficiency

For current object-state and airport current-overview retrieval, each returned state set R_i is compared with its manually established Gold set G_i . Across all query results, TP denotes correctly returned state instances, FP denotes returned instances absent from the Gold sets, and FN denotes Gold instances not returned. Precision, recall, and F1 are defined as

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad \text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (13)$$

Exact match additionally requires the complete returned set to equal the Gold set. Historical retrieval uses hit rate because multiple versions may satisfy the same object key and temporal conditions. Let v_i^* denote the manually identified target version for historical query i . The two query-level measures are

$$\text{Exact Match} = \frac{1}{N} \sum_{i=1}^N I(R_i = G_i), \quad \text{Historical Hit} = \frac{1}{N} \sum_{i=1}^N I(v_i^* \in R_i). \quad (14)$$

Exact match is applied to current object-state and airport current-overview retrieval. Precision, recall, and F1 are not reported for historical retrieval because the objective is to determine whether the manually identified version can be reached under the specified bitemporal conditions.

Query efficiency is evaluated using end-to-end latency. Each query is executed once for warm-up and then measured over five formal rounds. The five repetitions are averaged to obtain one sample-level latency, after which the mean, standard deviation, and 95th-percentile (P95) latency are calculated across the 500 query samples for each task–method pair. The 95% confidence interval of the mean is estimated using 2000 bootstrap resamples.

5.4.3. Ablation Metrics

The Current State Layer and State Version Layer are evaluated through functionally equivalent ablations. Before comparing latency and storage, semantic agreement verifies that Full BiTE and the ablated configuration return identical normalized object-state semantics. The normalization covers object identity, operational status, state payload, temporal validity, and source reference. Relative storage and relative mean latency express the corresponding storage and performance costs with Full BiTE normalized to $1.00\times$

$$\text{Semantic Agreement} = \frac{1}{N} \sum_{i=1}^N I(\text{Norm}(R_i^{\text{ablated}}) = \text{Norm}(R_i^{\text{full}})), \quad (15)$$

$$\text{Relative Storage} = \frac{S_{\text{ablated}}}{S_{\text{full}}}, \quad \text{Relative Mean Latency} = \frac{\bar{t}_{\text{ablated}}}{\bar{t}_{\text{full}}}, \quad (16)$$

where S denotes database storage and $\bar{t}$ denotes mean query latency. A relative storage value below $1.00\times$ indicates a storage saving, whereas a relative latency value above $1.00\times$ indicates additional query cost.

Lifecycle-maintenance ablation intentionally disables replacement, cancellation, and expiration processing and is therefore evaluated for retrieval correctness rather than functional equivalence. Precision, recall, and F1 follow Equation (13). The over-return rate and cumulative returned-instance count are defined as

$$\text{Over-return Rate} = \frac{FP}{TP + FP}, \quad \text{Cumulative Returns} = \sum_{i=1}^N |R_i|. \quad (17)$$

The over-return rate measures the proportion of returned instances that are obsolete or should not appear in the current-state result. Cumulative returns count all result occurrences across the sampled queries; if the same state is returned by different queries, each occurrence is retained without cross-query deduplication.

5.4.4. Auxiliary Query-Intent Metrics

The auxiliary natural-language experiment evaluates structured query-intent generation rather than database-state correctness or user-facing usability. LLM and rule-based outputs are evaluated at the field level [59]. Each predicted field–value pair is compared with its Gold value, and the resulting field-level true positives, false positives, and false negatives are aggregated across all questions to calculate micro precision, micro recall, and micro F1 using Equation (13).

Slot accuracy measures the proportion of intent fields whose predicted values exactly match the corresponding Gold values

$$\text{Slot Accuracy} = \frac{\sum_{i=1}^N \sum_{j \in \mathcal{S}_i} I(\hat{y}_{ij} = y_{ij}^*)}{\sum_{i=1}^N |\mathcal{S}_i|}, \quad (18)$$

where S_i is the set of intent fields for question i , $\hat{y}_{ij}$ is the predicted value, and y_{ij}^* is the Gold value. The evaluated fields include query scope, aerodrome, object type, target reference, status condition, business time, and system time. Field-level errors are further examined to identify conditions that are frequently omitted or incorrectly interpreted [60].

6. Results and Discussion

6.1. Full-Corpus Implementation

6.1.1. Database Construction Integrity

The full-corpus evaluation first examines whether the BiTE implementation generates the required object keys, state-version records, and traceability links. Table 4 reports these structural integrity checks together with the active projection statistics.

Table 4. Full-corpus database construction integrity.

Airport	Key Assignment Rate	Versioning Rate	Traceability-Link Rate	Active Projection Ratio
KBOS	99.99% (7092/7093)	100.00% (8857/8857)	100.00% (8857/8857)	62.23% (117/188)
KATL	100.00% (5246/5246)	100.00% (7353/7353)	100.00% (7353/7353)	60.71% (204/336)
KJFK	99.98% (4528/4529)	100.00% (5003/5003)	100.00% (5003/5003)	51.74% (134/259)
KCLT	99.98% (4153/4154)	100.00% (6006/6006)	100.00% (6006/6006)	37.04% (140/378)
KLAX	99.98% (4163/4164)	100.00% (4362/4362)	100.00% (4362/4362)	61.42% (156/254)
Overall	99.98% (25,182/25,186)	100.00% (31,581/31,581)	100.00% (31,581/31,581)	53.07% (751/1415)

Note: The key assignment rate is calculated over eligible notice-level construction cases. Versioning and traceability-link rates are calculated at the state-version level, while the active projection ratio is calculated at the object level. These indicators are internal implementation checks rather than measures of semantic correctness.

BiTE assigned object keys to 99.98% of the eligible construction cases and materialized all 31,581 expected state versions with complete event and source links. After lifecycle processing, 751 of the 1415 identified objects remained active, corresponding to an overall projection ratio of 53.07%. The airport-level variation reflects differences in lifecycle activity because many objects observed during the year were subsequently replaced, cancelled, expired, or closed.

These results confirm that the implementation generated the required database records and structural links. They do not establish whether the assigned keys and source relationships are semantically correct.

6.1.2. Lifecycle-Processing Statistics

The full-corpus lifecycle analysis examines the availability of explicitly referenced predecessor notices and the completeness of subsequent version generation, version closure, and current-state projection. Table 5 reports these implementation-level statistics.

Table 5. Full-corpus lifecycle-processing completeness.

Operation	Source Notices	Reference Resolution (Notice Level)	Version Generation (State-Version Level)	Version Closure (State-Version Level)	Projection Consistency (Update Level)
NOTAMN	22,544	–	100.00% (28,939/28,939)	–	100.00% (1214/1214)
NOTAMR	2642	99.43% (2627/2642)	100.00% (2642/2642)	99.19% (734/740)	100.00% (220/220)
NOTAMC	19,405	98.75% (19,162/19,405)	–	98.98% (23,235/23,474)	100.00% (1285/1285)
Overall	44,591	98.83% (21,789/22,047)	100.00% (31,581/31,581)	98.99% (23,969/24,214)	100.00% (2719/2719)

Note: Reference-resolution completeness denotes the proportion of NOTAMR and NOTAMC records whose explicitly referenced predecessor notices could be located within the corpus. Source notices and references are counted at the notice level; version generation and closure at the state-version level; and projection consistency at the object-state update level. Because one notice may affect multiple state versions, counts across columns are not directly comparable. A dash indicates that the operation is not applicable.

The 22,544 NOTAMN source notices produced 28,939 state versions because one notice could affect multiple aeronautical objects. Across all applicable cases, reference-resolution completeness was 98.83%, while version generation and closure completeness reached 100.00% and 98.99%, respectively. All 2719 evaluated projection updates were internally consistent.

The unresolved references were primarily associated with predecessor notices that were absent from or unavailable within the observation corpus. Accordingly, the reference-resolution values mainly reflect corpus availability and implementation completeness rather than the semantic accuracy of target matching. Object-level matching accuracy is evaluated separately against manually verified references in Section 6.2.

6.2. Independent Manual Validation

To complement the full-corpus implementation checks, BiTE was evaluated against the consensus-adjudicated manual references. The evaluation examined the semantic correctness of information-type mapping, canonical object-key construction, lifecycle resolution, state-version maintenance, and source attribution. Table 6 presents the results.

Table 6. Independent manual validation of event construction and state maintenance.

Evaluation Aspect	Unit	Manual Reference	Result
Information-type mapping	Source notice	Adjudicated information type	98.20% (982/1000)
Canonical object-key semantics	Object event	Adjudicated canonical object key	98.33% (1938/1971)
Object-level lifecycle alignment	Lifecycle decision	Verified target or termination relation	96.14% (796/828)
State-version creation	State version	Annotated expected state version	100.00% (1739/1739)
State-version closure	State version	Adjudicated closure outcome	98.16% (1707/1739)
Source attribution	State version	Adjudicated causative source NOTAM	100.00% (1739/1739)

Although normalization is treated as an upstream input-adaptation process, its type-routing and object-identification outputs were evaluated using the manually verified subset. BiTE correctly assigned 982 of the 1000 notice-level information types (98.20%). At the object-event level, 1938 of the 1971 canonical keys exactly matched the manual references (98.33%). These results provide independent evidence of the reliability of the normalized inputs used for subsequent state construction.

Most of the 33 object-key mismatches originated from incorrectly typed notices. Because the information type determines the applicable object schema and key-construction rules, a type error can propagate to the object type, identifier, and granularity components. One incorrectly typed notice may also produce multiple object events, explaining why the number of key mismatches exceeded the number of notice-level type errors. Object-key accuracy is therefore partly dependent on type-mapping reliability.

After notice-level lookup through the explicit reference identifier, BiTE correctly aligned 796 of the 828 manually verified lifecycle decisions with their object-level targets or termination relations, yielding an accuracy of 96.14%. This metric differs from the full-corpus reference-resolution completeness reported in Section 6.1: it evaluates whether the correct object-level event and state version were identified, rather than merely whether the referenced source notice was available. BiTE generated all 1739 expected state versions, while 1707 closure outcomes were correct. The 32 closure errors corresponded to the same 32 lifecycle-alignment errors, because a predecessor version could not be closed correctly when its object-level target was not established. Source attribution was correct for all 1739 evaluated versions.

The results reveal two principal error-propagation paths: information-type errors affect canonical object-key construction, whereas object-level lifecycle-alignment errors affect version closure. State-version creation and source attribution remained complete on

the manually verified sample. In contrast to the implementation-level checks in Section 6.1, these results provide independent evidence of semantic correctness.

6.3. Comparative Object-State Query Evaluation

To evaluate BiTE beyond internal database consistency, it was compared with three representative storage approaches: message-oriented MongoDB (Message-Mongo), relational bitemporal storage (SQL-Bitemporal), and an RDF-based NOTAM representation (RDF-NOTAM). All four implementations used the same source corpus and were evaluated using the same 1500 manually verified queries, comprising 500 current object-state queries, 500 airport current-overview queries, and 500 historical-state queries. Query results were normalized to a common object-state representation and evaluated against the manually established references rather than records generated by BiTE.

6.3.1. Query Correctness

The three tasks represent increasingly different access requirements: single-object retrieval evaluates precise current-state access, airport overview retrieval evaluates the completeness of an aggregated operational view, and historical retrieval evaluates whether a manually identified version can be recovered under specified temporal conditions. Table 7 reports the comparative correctness results.

Table 7. Object-state query correctness against manual references.

Method	Current-Object State				Airport Overview				Historical
	P	R	F1	Exact	P	R	F1	Exact	Hit
Message-Mongo	90.17%	90.90%	90.53%	89.40% (447/500)	86.38%	83.35%	84.84%	83.00% (415/500)	89.80% (449/500)
SQL-Bitemporal	97.90%	98.61%	98.25%	96.40% (482/500)	92.13%	93.18%	92.65%	91.80% (459/500)	94.40% (472/500)
RDF-NOTAM	92.28%	96.86%	94.51%	92.20% (461/500)	90.86%	92.33%	91.59%	90.80% (454/500)	92.20% (461/500)
Full BiTE	99.32%	99.54%	99.43%	99.00% (495/500)	98.31%	99.02%	98.66%	98.20% (491/500)	95.80% (479/500)

Note: Bold values indicate the best performance among all compared methods.

As illustrated in Figure 2, Full BiTE achieved the highest query-level correctness across all three tasks. For current object-state retrieval, it obtained an F1 score of 99.43% and an exact-match accuracy of 99.00%. SQL-Bitemporal was the strongest baseline, but its corresponding results were lower at 98.25% and 96.40%. The lower performance of Message-Mongo indicates that reconstructing object states directly from message-level records is more susceptible to incomplete lifecycle resolution and obsolete-state retention.

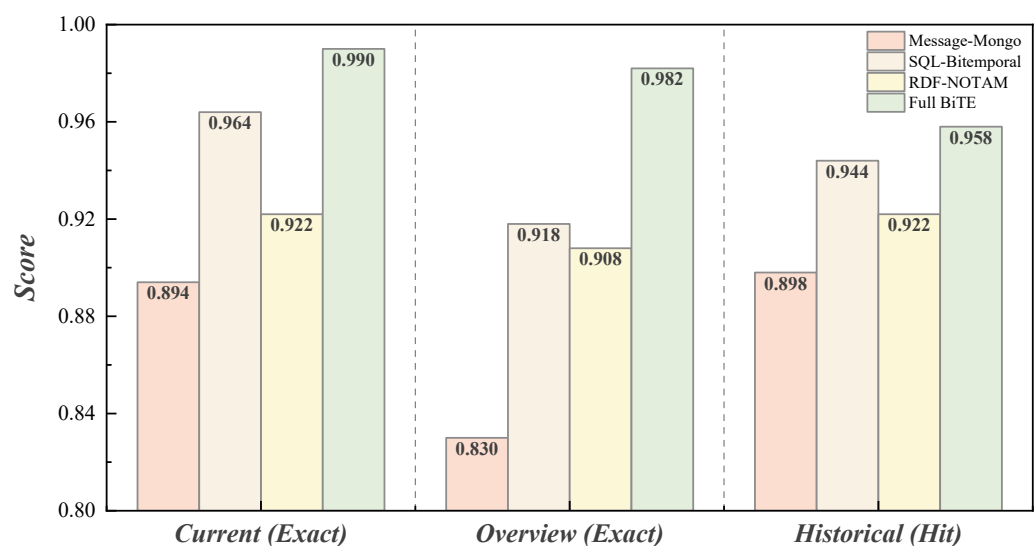


Figure 2. Comparison of query correctness across database implementations.

The advantage of BiTE was more pronounced for airport overview retrieval, where complete active-state sets must be assembled across multiple object types and lifecycle relations. BiTE achieved an F1 score of 98.66% and an exact-match accuracy of 98.20%, exceeding SQL-Bitemporal by 6.01 and 6.40 percentage points, respectively. This result supports the use of explicit object-level lifecycle maintenance and materialized current-state projections for aggregated operational views.

Historical retrieval remained the most challenging task. Full BiTE achieved the highest hit rate of 95.80% (479/500), followed by SQL-Bitemporal at 94.40%. Manual examination of the 21 missed BiTE cases identified three primary failure modes. Fourteen cases involved non-fixed or qualified expressions in Item C, including EST, PERM, and related end-time representations that could not always be mapped to a deterministic validity boundary. Five cases involved overlapping temporal intervals, which made selection among simultaneously eligible versions ambiguous. The remaining two cases resulted from insufficient object-key granularity, causing distinct operational scopes to share the same canonical key.

Thus, temporal normalization accounted for most historical retrieval failures, while overlapping-version selection and object-key granularity represented smaller but identifiable limitations. These cases could be addressed by explicitly representing estimated or permanent validity, defining deterministic precedence rules for overlapping versions, and incorporating additional segment- or facility-level attributes into canonical object keys.

6.3.2. Query Efficiency

Query efficiency was evaluated under the same workload and warm-cache conditions. To characterize both typical performance and response-time stability, Table 8 reports the mean latency with its 95% confidence interval, P95 latency, and standard deviation for each task–method combination.

Table 8. Object-state query latency.

Task	Method	Mean [95% CI] (ms)	P95 (ms)	SD (ms)
Current object-state	Message-Mongo	8.155 [7.563, 8.768]	23.264	7.002
	SQL-Bitemporal	9.932 [9.102, 10.773]	30.014	9.583
	RDF-NOTAM	10.613 [10.465, 10.769]	14.314	1.736
	Full BiTE	0.726 [0.710, 0.744]	1.053	0.198
Airport current overview	Message-Mongo	7.002 [5.853, 8.144]	43.192	13.262
	SQL-Bitemporal	7.759 [6.382, 9.245]	53.032	16.258
	RDF-NOTAM	9.902 [9.165, 10.626]	27.115	8.235
	Full BiTE	3.680 [3.502, 3.865]	13.704	2.019
Historical state	Message-Mongo	12.136 [11.581, 12.736]	24.757	6.670
	SQL-Bitemporal	17.512 [16.578, 18.488]	35.088	10.604
	RDF-NOTAM	10.417 [10.275, 10.558]	13.837	1.639
	Full BiTE	1.364 [1.310, 1.423]	2.882	0.651

Note: Each task comprised 500 queries. After one warm-up round, each query was measured five times and represented by its sample-level mean. Confidence intervals were estimated using 2000 bootstrap resamples. Bold values indicate the best query latency results among all compared methods.

Full BiTE achieved the lowest mean and P95 latency across all three tasks. Its mean latency remained below one millisecond for current object-state retrieval and below four milliseconds for both overview and historical retrieval. The largest improvements appeared in the current and historical tasks, where materialized projections and indexed state versions avoided repeated message-level reconstruction, relational temporal processing, or RDF graph-pattern evaluation.

Full BiTE also exhibited the lowest absolute latency dispersion in all three tasks, with standard deviations of 0.198 ms, 2.019 ms, and 0.651 ms for current, overview, and historical retrieval, respectively. The corresponding bootstrap confidence intervals were narrow,

indicating that the estimated mean latencies were stable across the sampled queries and repeated runs. The overview task showed greater variability than the other two tasks because the number and composition of active states differed across aerodromes. Nevertheless, its standard deviation and P95 latency remained substantially lower than those of the comparison methods. These results suggest that the materialized Current State Layer supports both efficient single-object access and relatively stable airport-level aggregation.

These results evaluate the combined data organization and query mechanism of each implementation rather than the database engines in isolation. In particular, Message-Mongo and Full BiTE both use MongoDB, yet their substantial differences in correctness, latency, and latency dispersion indicate that the observed gains primarily arise from object-level event decomposition, lifecycle-aware state maintenance, bitemporal version organization, and materialized current-state projections rather than from MongoDB alone.

6.4. Ablation Results

The ablation study examined the contributions of the Current State Layer, State Version Layer, and lifecycle-maintenance mechanism. For the two-layer ablations, the same lifecycle rules, bitemporal predicates, version-selection logic, and normalized output semantics were preserved. Current states were reconstructed from state versions after removing the Current State Layer, whereas historical states were reconstructed from event evidence after removing the State Version Layer. Semantic agreement with Full BiTE reached 100.00% (500/500) for each evaluated task, allowing the layer ablations to focus exclusively on latency and storage cost.

Table 9 reports the corresponding latency results under the same workload and measurement protocol as the comparative query experiment.

Table 9. Query latency under functionally equivalent layer ablation.

Task	Method	Mean [95% CI] (ms)	P95 (ms)	SD (ms)
Current object-state	Full BiTE	0.726 [0.710, 0.744]	1.053	0.198
	w/o Current State Layer	1.191 [1.092, 1.302]	3.538	1.229
Airport current overview	Full BiTE	3.680 [3.502, 3.865]	13.704	2.019
	w/o Current State Layer	23.189 [22.666, 23.735]	34.109	6.057
Historical state	Full BiTE	1.364 [1.310, 1.423]	2.882	0.651
	w/o State Version Layer	10.832 [9.814, 11.848]	41.798	11.561

Note: Bold values indicate the best query latency results among the compared methods.

Full BiTE achieved lower mean, P95, and standard-deviation values across all three tasks. Removing the Current State Layer had a relatively limited effect on single-object retrieval but substantially increased the cost of airport overview queries, which required multiple current states to be selected and aggregated from version histories. Similarly, reconstructing historical states from event evidence increased both latency and dispersion because the relevant events had to be replayed and temporally ordered at query time. The materialized layers therefore improved performance and stability without changing the returned object-state semantics.

To examine whether these improvements justify the additional storage, Table 10 relates the retained storage volume to the mean query latency of each configuration.

Removing the Current State Layer reduced total storage by only 3.19%, while increasing mean latency to $1.64\times$ for current object-state retrieval and $6.30\times$ for airport overview retrieval. The modest storage reduction therefore came at a substantial performance cost, particularly for aggregated current-state access. Removing the State Version Layer reduced storage by 18.66%, but increased historical retrieval latency to $7.94\times$. These results demon-

strate that the two materialized layers introduce measurable but limited storage overhead in exchange for substantially faster state access.

Table 10. Storage–latency trade-off under layer ablation.

Configuration	Task	Storage (MB)	Relative Storage	Relative Mean Latency
Full BiTE	All	113.168	1.00×	1.00×
w/o Current State Layer	Current	109.562	0.97×	1.64×
w/o Current State Layer	Overview	109.562	0.97×	6.30×
w/o State Version Layer	Historical	92.051	0.81×	7.94×

Note: Storage and latency are normalized to Full BiTE; latency ratios are calculated within the corresponding query task.

Unlike the functionally equivalent layer ablations, lifecycle-maintenance ablation deliberately disables the processing of replacement, cancellation, and expiration effects. It therefore evaluates the semantic role of lifecycle maintenance rather than its storage or latency contribution. Table 11 reports its effect on current-state retrieval.

Without lifecycle maintenance, recall remained at 100.00% because the valid current states were still included in the query results. Precision, however, decreased to 24.10%, and the over-return rate reached 75.90%, indicating that replaced, cancelled, and expired states were retained alongside the valid states. The increase in cumulative returns therefore represents obsolete result occurrences rather than additional valid or unique current states.

Table 11. Current-state retrieval under lifecycle-maintenance ablation.

Method	P	R	F1	Over-Return	Cumulative Returns
Full BiTE (reference)	100.00%	100.00%	100.00%	0.00%	7612
w/o Lifecycle Maintenance	24.10%	100.00%	38.84%	75.90%	31,581

Note: Cumulative returns count all result occurrences across 500 queries; repeated returns across different queries are retained.

Overall, the ablation results distinguish the roles of the three mechanisms. The Current State Layer provides an efficient materialized projection for current and aggregated access, while the State Version Layer supports direct and stable historical retrieval. These layers primarily determine the storage–latency trade-off, whereas lifecycle maintenance determines whether superseded states are correctly excluded from current operational views.

6.5. Auxiliary Query-Intent Parsing Results

This section reports the auxiliary evaluation of structured query-intent generation from informal bilingual operational questions. All results were calculated against the manually established Gold intents under the common experimental setting. Table 12 presents the comparative results.

Table 12. Auxiliary query-intent parsing performance.

Model	Micro Precision	Micro Recall	Micro F1	Slot Accuracy
Rule-based parser	0.5796	0.5524	0.5657	0.5369
Phi-4 Mini Instruct [57]	0.8851	0.7739	0.8257	0.8523
Llama 3.2 Instruct	0.8764	0.8015	0.8373	0.8677
GPT-5.4 nano	0.8511	0.9045	0.8770	0.8708
Gemma 3 Instruct [56]	0.9055	0.8668	0.8858	0.9046
Qwen3.5 Instruct	0.9514	0.8367	0.8904	0.8969
Qwen3-VL Instruct [58]	0.9299	0.8995	0.9144	0.9262
DeepSeek-V4-Flash	0.9349	0.9576	0.9461	0.9608

All LLM-based parsers substantially outperformed the rule-based baseline, indicating that fixed lexical rules were insufficient to cover the informal and heterogeneous question formulations. As illustrated in Figure 3, DeepSeek-V4-Flash achieved the best overall performance, with a Micro F1 of 0.9461 and slot accuracy of 0.9608, followed by Qwen3-VL Instruct. The precision–recall differences further indicate that some models tended to omit required conditions, whereas others introduced conditions not fully supported by the question.

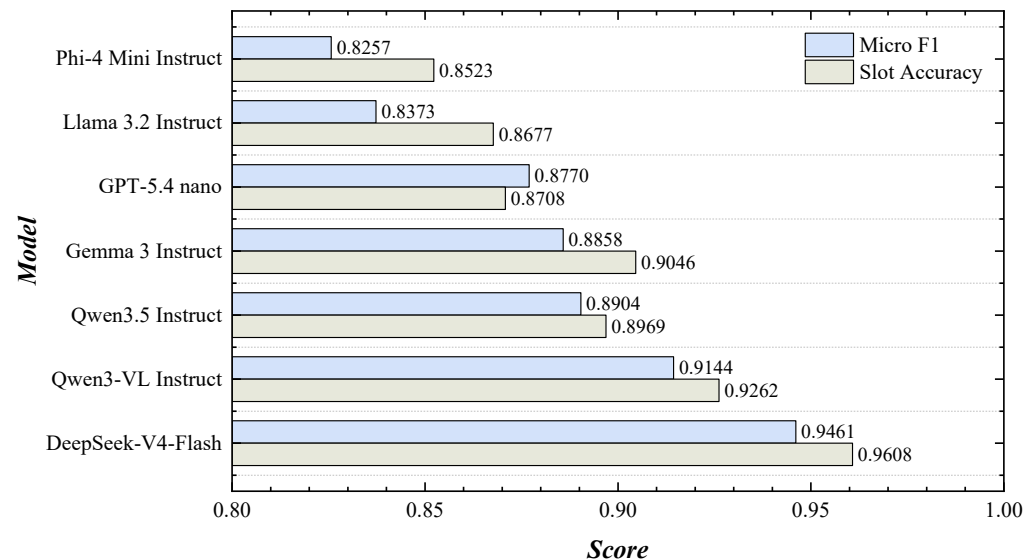


Figure 3. Performance comparison of LLM-based intent parsers.

Manual inspection showed that explicit aerodrome codes and clearly stated target references were generally identified reliably. The remaining errors were concentrated in mixed query scopes, implicit status expressions, and temporal conditions. Airport-wide questions emphasizing selected object categories were occasionally interpreted as single-type queries, while expressions such as “restricted” or “affected” could be mapped to overly specific status values. Informal and relative time expressions also increased the difficulty of historical-intent generation.

The results demonstrate the technical feasibility of mapping bilingual operational questions to the structured query schema of BiTE. This auxiliary experiment evaluates intent parsing rather than usability improvement over conventional querying; after intent generation, state selection and source tracing remain deterministic database operations. A comparative user study is left for future work.

6.6. General Discussion

Within the studied corpus and workload, BiTE outperformed the representative message-oriented, relational-bitemporal, and RDF-based implementations in object-state retrieval. Functionally equivalent ablations preserved query semantics but increased latency, showing that the materialized layers trade storage for access efficiency. By contrast, disabling lifecycle maintenance caused substantial over-return of obsolete states, confirming its contribution to current-state correctness.

The novelty of BiTE lies not in bitemporality, event sourcing, or versioning individually, but in coordinating them with object-level notice decomposition and NOTAMN, NOTAMR, and NOTAMC lifecycle semantics. BiTE complements AIXM and Digital NOTAM, which primarily support structured representation and exchange [5,16]; such structured records could also serve as inputs to the same state-maintenance framework.

MongoDB was selected as the engineering backend because its document model accommodates heterogeneous payloads and its compound indexes support multi-attribute bitemporal queries and direct access to materialized states. This choice is not an inherent constraint of BiTE, which could also be implemented using relational, wide-column, or graph-oriented databases. The experiments therefore assess the current implementation rather than claim that MongoDB is universally superior.

BiTE remains a research prototype rather than an operational system for safety-critical applications. Non-canonical temporal qualifiers, overlapping intervals, and insufficient object-key granularity can still propagate errors to downstream states. The query workload covers three core operational access patterns but does not include complex interval joins, cross-object analytical queries, or concurrent read–write workloads. Operational deployment would require broader external validation, uncertainty and exception handling, human oversight, and appropriate safety assurance. The natural-language experiment likewise evaluates only auxiliary intent-parsing feasibility.

7. Conclusions

This paper proposed BiTE, a bitemporal event-centered database framework that organizes normalized object-level events derived from textual NOTAMs into traceable state histories and current-state projections. Its contribution lies in coordinating event evidence, bitemporal versioning, materialized projection, and NOTAM-specific lifecycle semantics rather than in any of these mechanisms individually. A MongoDB-based prototype was evaluated using 44,591 NOTAMs from five U.S. aerodromes, independent manual references, representative database baselines, and functionally equivalent ablations. The results showed high but non-perfect semantic correctness and superior query performance under the studied workload. Layer ablations preserved normalized query semantics but increased latency, whereas disabling lifecycle maintenance caused substantial over-return of obsolete states. BiTE therefore provides a research-oriented foundation for object-state management rather than message-level NOTAM archiving.

Limitations and Future Work. The evaluation covers one year and five aerodromes and therefore does not establish generalizability across regions, regulatory environments, or longer operational periods. BiTE also depends on the availability and quality of its normalized inputs. When a referenced predecessor notice is absent or cannot be aligned with an object-level state, the lifecycle event is preserved as evidence but the corresponding version cannot be closed or replaced. Noisy Q-codes, ambiguous Item E descriptions, non-canonical temporal qualifiers, overlapping intervals, and insufficient object-key granularity may likewise propagate errors to state histories and query results.

MongoDB provides structural flexibility but does not eliminate semantic schema evolution. New information types, revised payload fields, or changes in object-key granularity may require versioned schemas, migration rules, and reconciliation of existing states. Integrating heterogeneous sources such as Digital NOTAM and AIXM may also introduce inconsistent identifiers, temporal conventions, duplicate updates, and conflicting state assertions. The current query workload covers three core operational access patterns but does not include complex interval joins, cross-object analytics, or concurrent read–write workloads.

Future work will examine schema versioning, source-priority and conflict-resolution rules, and uncertainty-aware temporal normalization, and will extend the evaluation to multi-region data, complex temporal and analytical queries, multi-airport aggregation, concurrent updates, and streaming access conditions. The present implementation has not been assessed for fault recovery, authoritative-data reconciliation, or aviation safety certification. BiTE should therefore be regarded as a research prototype; safety-critical

deployment would require human oversight, exception handling, comprehensive external validation, and the applicable assurance processes. The natural-language component remains an auxiliary intent-parsing experiment and would require separate usability and safety evaluation before operational use.

Author Contributions: Conceptualization, T.W. and X.L.; methodology, T.W. and X.L.; software, T.W.; validation, T.W., Y.L. and C.Z.; formal analysis, T.W. and Y.L.; investigation, T.W.; resources, X.L.; data curation, T.W.; writing—original draft preparation, T.W.; writing—review and editing, X.L., Y.L., C.Z. and R.K.; visualization, T.W.; supervision, X.L.; project administration, X.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Sichuan Provincial Science and Technology Program, grant number 2026YFHZ0286; the Fundamental Research Funds for the Central Universities, Civil Aviation Flight University of China, grant number 25CAFUC03046; and the Graduate Student Project of the Fundamental Research Funds for the Central Universities, Civil Aviation Flight University of China, grant number 26CAFUC10022.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data supporting the findings of this study are available from Tianyue Wei (vokal.will.cafuc@gmail.com) upon reasonable request. The raw NOTAM records are not publicly available because of data-access and redistribution restrictions.

Acknowledgments: The authors would like to thank the colleagues who provided valuable comments and technical assistance during the preparation of this study.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AIM	Aeronautical Information Management
AIP	Aeronautical Information Publication
AIRM	Air Traffic Management Information Reference Model
AIXM	Aeronautical Information Exchange Model
ATM	Air Traffic Management
BiTE	Bitemporal Event-Centered Database Framework
CNS	Communications, Navigation, and Surveillance
ICAO	International Civil Aviation Organization
JSON	JavaScript Object Notation
LLM	Large Language Model
NOTAM	Notice to Air Missions
NOTAMC	Cancellation NOTAM
NOTAMN	New NOTAM
NOTAMR	Replacement NOTAM
RDF	Resource Description Framework
SQL	Structured Query Language

Appendix A. Configuration of the Auxiliary Query-Intent Parsers

Table A1 lists the exact model identifiers, providers, and parameter scales used in the auxiliary experiment. All models were accessed on 17 June 2026. The open-weight models were deployed locally from their official checkpoints, whereas GPT-5.4 nano and DeepSeek-V4-Flash were accessed through their respective official APIs.

Table A1. Models used in the auxiliary query-intent experiment.

Model	Model Identifier	Provider	Scale
Rule-based parser	Custom deterministic rules	Local implementation	–
Phi-4 Mini Instruct	microsoft/Phi-4-mini-instruct	Microsoft/local	3.8B
Llama 3.2 Instruct	meta-llama/Llama-3.2-3B-Instruct	Meta/local	3.21B
GPT-5.4 nano	gpt-5.4-nano-2026-03-17	OpenAI API	Undisclosed
Gemma 3 Instruct	google/gemma-3-4b-it	Google/local	4B
Qwen3.5 Instruct	Qwen/Qwen3.5-4B	Alibaba Cloud/local	4B
Qwen3-VL Instruct	Qwen/Qwen3-VL-4B-Instruct	Alibaba Cloud/local	4B
DeepSeek-V4-Flash	deepseek-v4-flash	DeepSeek API	13B active

The open-weight models used their native chat templates with greedy decoding, temperature=0, top_p=1, do_sample=false, max_new_tokens=256, and a fixed seed of 2026. GPT-5.4 nano used reasoning_effort=none and max_output_tokens=256. DeepSeek-V4-Flash was evaluated in non-thinking mode with temperature=0, top_p=1, and max_tokens=256. Tools and function calling were disabled, and each question was evaluated once. Only text input was used for the multimodal models.

All LLMs received the same zero-shot semantic instructions, while their native chat templates were used to construct the model-specific input format. The common prompt followed the structure below.

You are a deterministic query-intent parser for the BiTE
aeronautical information database.

Convert one Chinese or English user question into exactly one
JSON object. Return JSON only. Do not provide explanations,
Markdown, or additional fields.

Output schema:

```
{
  "query_scope": "...",
  "aerodrome": "...",
  "object_type": null,
  "target_ref": null,
  "status_filter": null,
  "business_time": null,
  "system_time": null
}
```

Allowed query_scope values:

- "current_state": current status of a specified object, object type, or operational condition.
- "airport_overview": the complete current operational view of an aerodrome.
- "historical_state": the state of an object or aerodrome at a specified past time.

Allowed aerodrome values:

- "KATL", "KBOS", "KCLT", "KJFK", or "KLAX".

Allowed object_type values:

- "runway", "taxiway", "apron", "navaid", "procedure", "facility_service", "weather_surface", "maneuvering_area", "airspace", or null.

Allowed status_filter values:

- "CLSD", "U/S", "LIMITED", "OPEN", or null.

Temporal fields:

- business_time: the operational validity time expressed in ISO 8601 UTC format, or null.
- system_time: the database visibility or knowledge time expressed in ISO 8601 UTC format, or null.

Rules:

1. Use "historical_state" when the question explicitly refers to a past operational or database time.

2. Use "airport_overview" for airport-wide or multi-object current queries.
3. Otherwise, use "current_state".
4. Leave temporal fields null for current queries unless a specific time is explicitly stated.
5. Do not infer an object, status, or time that is unsupported by the question.
6. Use null for unspecified optional fields.
7. Return exactly the seven required keys.

User question:
{question}

The rule-based parser produced the same seven-field output structure. Aerodromes and target references were identified using dictionaries and regular expressions. Bilingual lexical rules were used to determine query scope, object type, and status, while explicit date and time expressions were normalized to ISO 8601 UTC values. Unresolved fields were assigned null values.

References

1. International Civil Aviation Organization. *Annex 15 to the Convention on International Civil Aviation: Aeronautical Information Services*, 16th ed.; International Civil Aviation Organization: Montreal, QC, Canada, 2018.
2. International Civil Aviation Organization. *Procedures for Air Navigation Services—Aeronautical Information Management (PANS-AIM)*, 1st ed.; International Civil Aviation Organization: Montreal, QC, Canada, 2018.
3. EUROCONTROL. *EUROCONTROL Guidelines: Operating Procedures AIS Dynamic Data (OPADD)*; Guidelines EUROCONTROL-GUID-121; EUROCONTROL: Brussels, Belgium, 2020.
4. EUROCONTROL; Federal Aviation Administration. *Digital NOTAM Event Specification*, Version 1.0. 2011. Available online: https://aixm.aero/sites/default/files/imce/library/Digital_NOTAM_Spec/digital_notam_event_specification_1.0.doc (accessed on 19 July 2026).
5. Aeronautical Information Exchange Model. Digital NOTAM. Available online: <https://aixm.aero/page/digital-notam> (accessed on 19 July 2026).
6. Gringinger, E. Ontology-based representation and semantic querying of digital notices to airmen. In *Proceedings of the 2014 Integrated Communications, Navigation and Surveillance Conference (ICNS) Conference Proceedings*, Herndon, VA, USA, 8–10 April 2014; pp. 1–23. [\[CrossRef\]](#)
7. Burgstaller, F.; Steiner, D.; Schrefl, M.; Gringinger, E.; Wilson, S.; van der Stricht, S. AIRM-based, fine-grained semantic filtering of notices to airmen. In *Proceedings of the 2015 Integrated Communication, Navigation and Surveillance Conference (ICNS)*, Herndon, VA, USA, 21–23 April 2015; pp. D3–1–D3–13. [\[CrossRef\]](#)
8. Baigang, M.; Yi, F. A review: development of named entity recognition (NER) technology for aeronautical information intelligence. *Artif. Intell. Rev.* **2022**, *56*, 1515–1542. [\[CrossRef\]](#)
9. Arnold, A.; Ernez, F.; Kobus, C.; Martin, M.C. Knowledge extraction from aeronautical messages (NOTAMs) with self-supervised language models for aircraft pilots. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*, Seattle, WA, USA, 10–15 July 2022; Association for Computational Linguistics: Stroudsburg, PA, USA, 2022; pp. 188–196. [\[CrossRef\]](#)
10. Liu, M.; Fang, Q.; Yang, Y.; Zhao, C.; Cai, K. Knots: A large-scale multi-agent enhanced expert-annotated dataset and LLM prompt optimization for NOTAM semantic parsing. *Adv. Eng. Inform.* **2026**, *69*, 104086. [\[CrossRef\]](#)
11. Lai, X.; Zeng, J.; Dai, Y.; Han, S. A Spatio-Temporal Resource Description Framework Schema Model for Aeronautical Dynamic Information Based on Semantic Analysis. *J. Inf. Technol. Res.* **2022**, *15*, 1–14. [\[CrossRef\]](#)
12. Dong, B.; Luo, C.; Hao, K.; Liu, A.; Li, X. Automated Processing Method for Chinese NOTAMs Based on Knowledge Graph. *J. Aerosp. Inf. Syst.* **2024**, *21*, 906–913. [\[CrossRef\]](#)
13. AIXM Change Control Board. *AIXM Version 5.1.1 Temporality Concept*. EUROCONTROL and Federal Aviation Administration, Version 1.1. 2019. Available online: <https://aixm.aero/page/aixm-511-specification> (accessed on 19 July 2026).
14. Jensen, C.S.; Soo, M.D.; Snodgrass, R.T. Unifying temporal data models via a conceptual model. *Inf. Syst.* **1994**, *19*, 513–547. [\[CrossRef\]](#)
15. Jensen, C.S.; Snodgrass, R.T. Semantics of time-varying information. *Inf. Syst.* **1996**, *21*, 311–352. [\[CrossRef\]](#)
16. EUROCONTROL. Aeronautical Information Exchange Model (AIXM), n.d. Available online: <https://www.eurocontrol.int/model/aeronautical-information-exchange-model> (accessed on 19 July 2026).
17. Tanguy, L.; Tulechki, N.; Urieli, A.; Hermann, E.; Raynal, C. Natural language processing for aviation safety reports: From classification to interactive analysis. *Comput. Ind.* **2016**, *78*, 80–95. [\[CrossRef\]](#)

18. Zhao, Y.; Liu, H.; Chen, Z. Named Entity Recognition for Chinese Aviation Security Incident Based on BiLSTM and CRF. In Proceedings of the 2021 2nd Asia Conference on Computers and Communications (ACCC), Singapore, 24–27 September 2021; pp. 89–94. [\[CrossRef\]](#)
19. Dong, B.; Luo, C.; Liu, A.; Li, X. Notice-to-Airmen Text-Error-Correction Method Based on Ccf-MacBERT and Knowledge Graph. *J. Aerosp. Inf. Syst.* **2025**, *22*, 145–153. [\[CrossRef\]](#)
20. Mi, B.; Fan, Y.; Sun, Y. NOTAM Text Analysis and Classification Based on Attention Mechanism. *J. Phys. Conf. Ser.* **2022**, *2171*, 012042. [\[CrossRef\]](#)
21. Keller, R.M.; Ranjan, S.; Wei, M.Y.; Eshow, M.M. Semantic representation and scale-up of integrated air traffic management data. In Proceedings of the International Workshop on Semantic Big Data, San Francisco, CA, USA, 26 June–1 July 2016; pp. 1–6. [\[CrossRef\]](#)
22. Morărașu, M.M.; Horațiu Roman, C. AI-driven Optimization of Operational NOTAM Management. In Proceedings of the 2024 Integrated Communications, Navigation and Surveillance Conference (ICNS), Herndon, VA, USA, 23–25 April 2024; pp. 1–6. [\[CrossRef\]](#)
23. Salyer, O.; Phelps, B.M.; Mourning, C. Automated Visualization of NOTAMs with Natural Language Processing. In Proceedings of the AIAA SCITECH 2025 Forum. American Institute of Aeronautics and Astronautics, Orlando, FL, USA, 6–10 January 2025. [\[CrossRef\]](#)
24. Neelamegam, G.; Bhogal, J.; Samimi, P.; Ozturkoglu, O. Comprehensive insights into bitemporal databases: A PRISMA-guided systematic literature review. *J. Data, Inf. Manag.* **2026**, *8*, 71–96. [\[CrossRef\]](#)
25. Böhlen, M.H.; Dignös, A.; Gamper, J.; Jensen, C.S. Temporal Data Management: An Overview. In *Business Intelligence and Big Data*; Zimányi, E., Ed.; Springer International Publishing: Cham, Switzerland, 2018; pp. 51–83. [\[CrossRef\]](#)
26. Kulkarni, K.; Michels, J.E. Temporal features in SQL:2011. *ACM SIGMOD Rec.* **2012**, *41*, 34–43. [\[CrossRef\]](#)
27. Kumar, A.; Tsotras, V.; Faloutsos, C. Designing access methods for bitemporal databases. *IEEE Trans. Knowl. Data Eng.* **1998**, *10*, 1–20. [\[CrossRef\]](#)
28. Salzberg, B.; Tsotras, V.J. Comparison of access methods for time-evolving data. *ACM Comput. Surv.* **1999**, *31*, 158–221. [\[CrossRef\]](#)
29. Meanrach, W.; Chittayasothorn, S. A Bitemporal Multilevel Secure Database System. In Proceedings of the AFRICON 2007, Windhoek, South Africa, 26–28 September 2007; pp. 1–7. [\[CrossRef\]](#)
30. Kaufmann, M.; Fischer, P.M.; May, N.; Ge, C.; Goel, A.K.; Kossmann, D. Bi-temporal Timeline Index: A data structure for Processing Queries on bi-temporal data. In Proceedings of the 2015 IEEE 31st International Conference on Data Engineering, Seoul, Republic of South Korea, 13–17 April 2015; pp. 471–482. [\[CrossRef\]](#)
31. Lomet, D.; Barga, R.; Mokbel, M.; Shegalov, G. Transaction Time Support Inside a Database Engine. In Proceedings of the 22nd International Conference on Data Engineering (ICDE'06), Atlanta, GA, USA, 3–7 April 2006; p. 35. [\[CrossRef\]](#)
32. Stantic, B.; Sattar, A.; Terenziani, P. The POINT approach to represent now in bitemporal databases. *J. Intell. Inf. Syst.* **2008**, *32*, 297–323. [\[CrossRef\]](#)
33. Wu, D.; Wang, H.T.; Tansel, A.U. A survey for managing temporal data in RDF. *Inf. Syst.* **2024**, *122*, 102368. [\[CrossRef\]](#)
34. Gutierrez, C.; Hurtado, C.; Vaisman, A. Temporal RDF. In *Proceedings of the The Semantic Web: Research and Applications*; Gómez-Pérez, A., Euzenat, J., Eds.; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2005; pp. 93–107. [\[CrossRef\]](#)
35. Gutierrez, C.; Hurtado, C.; Vaisman, A. Introducing Time into RDF. *IEEE Trans. Knowl. Data Eng.* **2007**, *19*, 207–218. [\[CrossRef\]](#)
36. Tansel, A.U.; Wu, D.; Wang, H.T. Time Travel with the BiTemporal RDF Model. *Mathematics* **2025**, *13*, 2109. [\[CrossRef\]](#)
37. Wu, D.; Wang, H.T.; Tansel, A.U. On the Implementations of the BiTemporal RDF Model: An Experimental Approach. *Informatics* **2026**, *13*, 61. [\[CrossRef\]](#)
38. Katreddy, S.S. Event-Driven Cloud Architectures for Real-Time Data Processing. *Econ. Sci.* **2017**, *13*. [\[CrossRef\]](#)
39. Mareco, C.A.; Bertossi, L. Specification and Implementation of Temporal Databases in a Bitemporal Event Calculus. In *Advances in Conceptual Modeling*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 74–85. [\[CrossRef\]](#) [\[PubMed\]](#)
40. Gańczarski, S. Database Versions to Represent Bitemporal Databases. In *Database and Expert Systems Applications*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 832–841. [\[CrossRef\]](#) [\[PubMed\]](#)
41. Wondoh, J.; Grossmann, G.; Gasevic, D.; Reichert, M.; Schrefl, M.; Stumptner, M. Bitemporal Support for Business Process Contingency Management. In *Advances in Conceptual Modeling*; Springer International Publishing: Berlin/Heidelberg, Germany, 2015; pp. 109–118. [\[CrossRef\]](#) [\[PubMed\]](#)
42. Wondoh, J.; Grossmann, G.; Stumptner, M. Utilising bitemporal information for business process contingency management. In Proceedings of the ACSW '16 Australasian Computer Science Week Multiconference, Canberra, Australia, 1–5 February 2016; pp. 1–10. [\[CrossRef\]](#)
43. Anselma, L.; Ballerini, M.; Giordano, P.; Raina, E.; Terenziani, P. Bitemporal Property Graphs: Dealing with Both Valid and Transaction Time. In *Advances in Databases and Information Systems*; Springer Nature: Cham, Switzerland, 2025; pp. 234–249. [\[CrossRef\]](#)

44. Thompson, R.; van Oosterom, P. Bi-temporal foundation for LADM v2: Fusing event and state based modelling of Land administration data 2D and 3D. *Land Use Policy* **2021**, *102*, 105246. [\[CrossRef\]](#)
45. Carroll, J.J.; Bizer, C.; Hayes, P.; Stickler, P. Named Graphs, Provenance and Trust. In Proceedings of the WWW '05 14th International Conference on World Wide Web, Chiba, Japan, 10–14 May 2005; pp. 613–622. [\[CrossRef\]](#)
46. Taelman, R.; Vander Sande, M.; Van Herwegen, J.; Mannens, E.; Verborgh, R. Triple Storage for Random-Access Versioned Querying of RDF Archives. *SSRN Electron. J.* **2018**. Available online: <https://ssrn.com/abstract=3248501> (accessed on 19 July 2026). [\[CrossRef\]](#)
47. Pelgrin, O.; Taelman, R.; Galárraga, L.; Hose, K. GLENDa: Querying RDF Archives with Full SPARQL. In *The Semantic Web: ESWC 2023 Satellite Events*; Springer Nature: Cham, Switzerland, 2023; pp. 75–80. [\[CrossRef\]](#)
48. Gao, Y.; Chen, X.; Du, X. A Big Data Provenance Model for Data Security Supervision Based on PROV-DM Model. *IEEE Access* **2020**, *8*, 38742–38752. [\[CrossRef\]](#)
49. Eshtay, M.; Sleit, A.; Aldwairi, M. Implementing Bi-Temporal Properties into Various NoSQL Database Categories. *Int. J. Comput.* **2019**, *18*, 45–52. [\[CrossRef\]](#)
50. Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv* **2019**, arXiv:1907.11692.
51. Yan, L.; Zhang, Z.; Yang, D. Temporal RDF(S) Data Storage and Query with HBase. *J. Comput. Inf. Technol.* **2020**, *27*, 17–30. [\[CrossRef\]](#)
52. Motik, B. Representing and Querying Validity Time in RDF and OWL: A Logic-Based Approach. *J. Web Semant.* **2012**, *12–13*, 3–21. [\[CrossRef\]](#)
53. Katsogiannis-Meimarakis, G.; Koutrika, G. A Survey on Deep Learning Approaches for Text-to-SQL. *VLDB J.* **2023**, *32*, 905–936. [\[CrossRef\]](#)
54. Fleiss, J.L. Measuring Nominal Scale Agreement among Many Raters. *Psychol. Bull.* **1971**, *76*, 378–382. [\[CrossRef\]](#)
55. Landis, J.R.; Koch, G.G. The Measurement of Observer Agreement for Categorical Data. *Biometrics* **1977**, *33*, 159–174. [\[CrossRef\]](#)
56. Gemma Team. Gemma 3 Technical Report. *arXiv* **2025**, arXiv:2503.19786.
57. Microsoft. Phi-4-Mini Technical Report: Compact yet Powerful Multimodal Language Models via Mixture-of-LoRAs. *arXiv* **2025**, arXiv:2503.01743.
58. Bai, S.; Cai, Y.; Chen, R.; Chen, K.; Chen, X.; Cheng, Z.; Deng, L.; Ding, W.; Gao, C.; Ge, C.; et al. Qwen3-VL Technical Report. *arXiv* **2025**, arXiv:2511.21631.
59. Liu, Y.; Li, D.; Wang, K.; Xiong, Z.; Shi, F.; Wang, J.; Li, B.; Hang, B. Are LLMs Good at Structured Outputs? A Benchmark for Evaluating Structured Output Capabilities in LLMs. *Inf. Process. Manag.* **2024**, *61*, 103809. [\[CrossRef\]](#)
60. Weld, H.; Huang, X.; Long, S.; Poon, J.; Han, S.C. A Survey of Joint Intent Detection and Slot Filling Models in Natural Language Understanding. *ACM Comput. Surv.* **2022**, *55*, 1–38. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.