International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

Hybrid Neural-Symbolic Natural Language Understanding with Deterministic Routing for Multi-Intent Command Processing

K Hari, S D Tenith Raj, Dr. N. Devakirubai

UG Student, Department of Artificial Intelligence and Data Science, RP Sarathy Institute of Technology, Poosaripatti, Salem, Tamil Nadu, India

UG Student, Department of Artificial Intelligence and Data Science, RP Sarathy Institute of Technology, Poosaripatti, Salem, Tamil Nadu, India

Assistant Professor, Department of Artificial Intelligence and Data Science, RP Sarathy Institute of Technology, Poosaripatti, Salem, Tamil Nadu, India

Publication History: Received: 25.02.2026; Revised: 20.03.2026; Accepted: 25.03. 2026; Published: 28.03.2026.

ABSTRACT: Natural language understanding (NLU) systems used in command-driven interfaces must process both simple and multi-intent queries while maintaining high accuracy and low latency. Conventional approaches typically rely on a single model architecture, which can lead to inefficiencies when queries vary significantly in complexity. To address this limitation, this paper proposes a hybrid neural–symbolic NLU framework that integrates deterministic routing, machine learning (ML), deep learning (DL), slot extraction, and knowledge graph validation.

The proposed system employs a deterministic complexity-aware routing mechanism that directs low-complexity queries to a lightweight ML ensemble classifier, while complex or multi-intent queries are processed by a transformer-based DL model for deeper semantic analysis. The predicted intents are then passed to a sequence-to-structure slot extraction module based on the T5 architecture to identify structured parameters required for command execution. A symbolic knowledge graph layer further performs intent compatibility checks, dependency validation, and slot constraint verification to ensure logical consistency in the final interpretation.

Experimental evaluation demonstrates that the proposed hybrid architecture outperforms ML-only and DL-only baselines. The system achieves 98.0% intent accuracy and 95.9% slot accuracy while maintaining an average end-to-end latency of 333.4 ms through efficient routing. These results indicate that integrating neural learning with symbolic validation and deterministic routing provides an effective and scalable solution for multi-intent command understanding in offline intelligent agents.

KEYWORDS: Hybrid Natural Language Understanding, Neural–Symbolic AI, Multi-Intent Detection, Deterministic Routing, Slot Filling, Knowledge Graph Validation, Command Understanding.

I. INTRODUCTION

Natural language understanding (NLU) systems are a core component of command-driven interfaces, where user utterances must be converted into structured representations for execution [1, 2]. Significant progress has been made in intent detection and slot filling through joint modeling approaches [3, 4]. However, most existing methods assume that each input corresponds to a single intent and rely on a fixed model architecture for processing [5, 6]. In practice, real-world command interactions frequently involve multiple intents within a single query, such as executing several system operations simultaneously [7, 8]. These scenarios introduce challenges related to semantic complexity, computational efficiency, and logical consistency [9, 10].

Existing NLU approaches exhibit fundamental limitations when applied to such settings. Traditional machine learning models provide low-latency inference but lack the capacity to capture complex semantic relationships required for multi-intent understanding [10, 20]. In contrast, deep learning models, particularly transformer-based architectures,

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

achieve strong performance in multi-intent scenarios but incur substantial computational overhead when applied uniformly to all queries [7, 9]. Joint intent–slot models improve semantic representation [3, 4] but still operate under a single-model paradigm, making them inefficient for handling queries of varying complexity. Furthermore, purely neural approaches lack mechanisms to enforce logical consistency [22, 25], often producing predictions that violate domain constraints or represent incompatible actions.

These limitations highlight a critical gap in current NLU systems. While prior work has explored multi-intent modeling [5–8], efficient inference architectures [17, 18], and neural–symbolic reasoning [22, 25], no existing framework jointly addresses query complexity adaptation, multi-intent understanding, and logical validation within a unified pipeline. This gap becomes particularly significant in real-time command-driven systems, where both accuracy and latency are critical requirements.

To address this challenge, this paper proposes a hybrid neural–symbolic NLU framework that integrates deterministic routing, machine learning and deep learning models, neural slot extraction, and knowledge graph validation. The system employs a complexity-aware routing mechanism to direct simple queries to lightweight machine learning models and complex or multi-intent queries to transformer-based models. The predicted intents are further processed by a sequence-to-structure slot extraction model, and a symbolic knowledge graph layer performs post-hoc validation to ensure logical consistency of the final output.

This study is guided by the following research questions:

- RQ1: How can query complexity be effectively modeled for adaptive routing in NLU systems?
- RQ2: To what extent does hybrid routing improve latency while maintaining intent prediction accuracy?
- RQ3: How does symbolic validation impact the logical consistency of multi-intent predictions?

The primary contributions of this paper are as follows:

- A hybrid neural–symbolic NLU architecture that integrates deterministic routing with neural intent prediction and symbolic validation.
- A complexity-aware routing mechanism that reduces unnecessary computational overhead by selectively applying deep models only to complex queries.
- A post-hoc knowledge graph validation framework that enforces logical consistency without modifying the underlying neural models.
- A comprehensive evaluation demonstrating improvements in both accuracy and latency compared to single-model baselines.

The remainder of this paper is structured to systematically address the research questions outlined above. Section 2 reviews existing approaches and critically analyzes their limitations in handling multi-intent queries, computational efficiency, and logical consistency. Section 3 presents the proposed hybrid neural–symbolic architecture, detailing the design of the deterministic routing mechanism, neural intent models, and symbolic validation layer. Section 4 describes the dataset construction and experimental setup used to evaluate the system. Section 5 reports the experimental results, analyzing the effectiveness of the proposed approach in terms of accuracy, latency, and logical consistency. Finally, Section 6 concludes the paper and discusses potential directions for future research.

II. LITERATURE REVIEW

Natural language understanding (NLU) plays a fundamental role in task-oriented dialogue systems and intelligent assistants, where user utterances must be transformed into structured semantic representations. Two primary subtasks are typically involved: intent detection, which identifies the user's objective, and slot filling, which extracts relevant parameters from the utterance. Traditional NLU pipelines treated these tasks independently, often resulting in error propagation and limited contextual interaction between the tasks. As research progressed, joint modeling approaches were introduced to capture the inherent relationship between intents and slots, leading to improved performance and more coherent semantic interpretation. This section reviews existing work across several major directions, including joint intent–slot modeling, multi-intent natural language understanding, efficient NLU architectures, and neural–symbolic reasoning approaches.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

2.1 Joint Intent Detection and Slot Filling Architectures

Early studies in NLU largely addressed intent detection and slot filling as separate tasks, where a classification model predicted the user intent while a sequence labeling model identified slot values. However, this pipeline architecture often suffered from cascading errors because the output of one component directly influenced the other. To address this limitation, researchers began developing joint learning models that simultaneously predict intents and slots using shared representations.

Several neural network architectures have been proposed for this purpose. Pretrained transformer-based approaches demonstrated that shared semantic encoders can improve both intent detection and slot labeling by capturing contextual dependencies across tokens. For instance, joint transformer models integrate intent classification and sequence labeling within a unified architecture, allowing the model to exploit correlations between the two tasks [9]. Similarly, hybrid deep learning architectures combining convolutional neural networks and bidirectional long short-term memory networks have been explored to capture both local and sequential contextual features within utterances [10].

Another important line of research focuses on hierarchical modeling of semantic structures. Capsule-based neural architectures explicitly model hierarchical relationships among words, slots, and intents, enabling the system to capture semantic dependencies across different representation levels [11]. In addition, multitask learning frameworks have been proposed to encourage information sharing between intent detection and slot filling modules. Some studies further incorporate external knowledge sources such as knowledge bases to enrich semantic representations and mitigate the limitations of purely data-driven learning [12]. Although these approaches improve semantic modeling compared with traditional pipeline methods, most of them assume that each utterance expresses a single intent, which limits their applicability in more complex real-world scenarios.

2.2 Multi-Intent Natural Language Understanding

In practical conversational systems, users frequently express multiple intents within a single utterance. For example, a user may ask to check the weather and schedule a meeting in one query. Handling such cases requires models capable of identifying multiple intents while simultaneously extracting associated slot values. Consequently, multi-intent natural language understanding has become an active research area.

Graph-based interaction models represent one of the most influential approaches for multi-intent NLU. These models construct interaction graphs that capture relationships among intents and slots, allowing information to propagate between them during prediction. Non-autoregressive graph-based architectures have been proposed to enable efficient parallel decoding while preserving semantic interactions between intents and slot representations [13]. Similarly, dynamic graph frameworks model complex dependencies between slot tokens and intent labels, enabling the system to adaptively update semantic relationships during inference [14]. Domain-specific frameworks have also been explored, particularly in automotive conversational assistants, where heterogeneous interaction networks are used to manage multi-intent commands in real-time environments [15].

More recent research focuses on attention mechanisms and advanced representation learning techniques to improve multi-intent modeling. Co-attention architectures have been introduced to explicitly model bidirectional interactions between intents and slots without relying on explicit graph structures [16]. Label-aware attention networks further enhance semantic modeling by integrating contrastive learning strategies to improve representation quality and generalization in complex multi-intent scenarios [17].

Contrastive learning has also been applied to multi-intent detection tasks to improve representation learning in high-dimensional semantic spaces. Prototypical contrastive learning frameworks learn multiple semantic representations for each utterance under different intent contexts, enabling the model to capture nuanced semantic relationships between user queries and intent labels [18]. Similarly, contrastive task specialization methods use sentence encoder fine-tuning strategies to improve multi-label intent classification and mitigate semantic ambiguity [19]. Despite these advances, accurately capturing the complex relationships between multiple intents and their associated slots remains a challenging problem.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

2.3 Efficient Architectures for Natural Language Understanding

As NLU systems are increasingly deployed in real-world applications such as mobile devices and embedded assistants, computational efficiency has become an important design consideration. Large transformer models provide strong performance but often require substantial computational resources and latency, which can limit their usability in real-time environments.

To address this challenge, researchers have proposed lightweight transformer architectures and model compression techniques. Mobile-oriented transformer models employ bottleneck structures and parameter sharing mechanisms to reduce model size while maintaining competitive performance [20]. Similarly, knowledge distillation techniques have been used to compress large pretrained models into smaller architectures suitable for deployment on resource-constrained devices [21].

Another approach focuses on modular architectures that dynamically allocate computational resources during inference. Mixture-of-experts frameworks introduce routing mechanisms that select a subset of specialized model components for each input, improving computational efficiency without sacrificing model capacity [22]. Lightweight on-device intent classifiers have also been developed using character-level representations to support efficient inference in embedded environments [23].

In addition, large-scale industrial NLU systems often employ semi-supervised learning techniques to leverage massive amounts of unlabeled data. Production pipelines using teacher–student learning frameworks and pseudo-labeling strategies have demonstrated substantial performance improvements while reducing the need for expensive human annotation [24]. These developments highlight the importance of balancing accuracy and efficiency in modern NLU system design.

2.4 Neural–Symbolic Natural Language Understanding

While deep neural networks have achieved impressive performance in NLU tasks, purely neural approaches sometimes struggle with logical reasoning and consistency verification. To address this limitation, researchers have explored neural–symbolic frameworks that integrate neural learning with symbolic reasoning mechanisms.

Neural–symbolic systems combine statistical learning models with symbolic knowledge representations to enable reasoning over structured knowledge. Such architectures typically employ neural components for perception and representation learning, while symbolic modules perform logical inference and consistency validation [25]. In the context of NLU, knowledge graphs and external knowledge bases have been incorporated to provide additional semantic context and improve prediction reliability.

Several studies have proposed joint models that integrate knowledge graphs into intent detection and slot filling tasks. By leveraging external knowledge sources, these models can capture semantic relationships that may not be easily learned from limited training data alone [26]. Multitask learning approaches have also incorporated knowledge bases to enhance shared representations and improve the semantic interaction between intent detection and slot filling modules [27]. These neural–symbolic approaches demonstrate that combining data-driven learning with structured knowledge can enhance both interpretability and robustness in NLU systems.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

2.5 Limitations of Existing Approaches and Research Gap

Table 1: Comparative Analysis of Multi-Intent NLU Approaches

Model /Author	Method Type	Core Technique	Multi-Intent	Slot Handling	Efficiency	Limitations
CNN-BiLSTM (Wang et al.)	Hybrid DL	Combines CNN for feature extraction and BiLSTM for sequence modeling	Partial	Joint	Medium	Limited ability to capture long-range dependencies and multi-intent interactions
Capsule-NLU (Zhang et al.)	Capsule Network	Uses capsule layers to model hierarchical relationships between intent and slots	Yes	Joint	Low	High computational complexity and difficult training convergence
AGIF (Qin et al.)	Graph-based	Models intent-slot dependencies using graph interaction networks	Yes	Joint	Low	Graph construction overhead and limited scalability
Co-Interactive Model	Graph-based	Uses bidirectional interaction between intent and slot representations	Yes	Joint	Low	Complex architecture with high inference cost
MISCA (Pham et al.)	Attention-based	Applies co-attention and label attention for intent-slot interaction	Yes	Joint	Medium	Computationally intensive and lacks explicit constraint modeling
GL-GIN	Graph + Transformer	Non-autoregressive graph model for multi-intent detection	Yes	Joint	Medium	Increased architectural complexity and resource requirements
TFMN (Chen et al.)	Transformer	Predicts intent labels and count without threshold tuning	Yes	Joint	Low	High latency and limited interpretability
Self-Distillation	Joint learning	Uses mutual learning between intent and slot representations	Yes	Joint	Low	Multi-stage training increases complexity
LCAN	Contrastive	Label-aware contrastive attention for intent representation	Yes	Partial	Medium	Focuses more on representation than full pipeline modeling
PCMID	Contrastive	Uses prototypical contrastive learning for intent classification	Yes	Partial	Medium	Limited handling of structured slot extraction
Neural-Symbolic NLU	Hybrid	Combines neural models with symbolic knowledge constraints	Yes	Partial	Medium	Does not address model selection or latency optimization
Proposed Method	Hybrid (Ours)	Routing-based ML + DL + T5 slot extraction + KG validation	Yes	Structured	High	Increased system integration complexity

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

Despite significant progress in NLU research, several limitations remain. First, many existing models rely on a single architectural paradigm, such as purely neural or graph-based approaches, which may not be optimal for handling the diverse complexity of user queries. Second, although multi-intent detection models have improved semantic modeling, they often struggle to maintain logical consistency between predicted intents and slot values. Third, high-performing deep learning models typically require substantial computational resources, which can limit their applicability in real-time systems.

Furthermore, neural models generally lack mechanisms for validating predictions against structured knowledge, which can result in logically inconsistent interpretations. Although neural–symbolic approaches partially address this issue, they are rarely integrated with efficient inference architectures that can adapt to varying query complexity.
To provide a comprehensive overview of existing multi-intent NLU methods, Table 1 summarizes representative approaches, their core techniques, and limitations.

To address these limitations, this work proposes a hybrid neural–symbolic NLU framework that combines deterministic routing, machine learning and deep learning models, structured slot extraction, and knowledge graph validation. By dynamically routing queries based on complexity and incorporating symbolic reasoning mechanisms, the proposed system aims to achieve improved accuracy, efficiency, and logical consistency in multi-intent command understanding.

III. METHODOLOGY

This section describes the design and operational workflow of the proposed hybrid neural–symbolic natural language understanding (NLU) framework. The architecture integrates deterministic routing, machine learning and transformer-based intent classification models, sequence-to-structure slot extraction, symbolic knowledge graph validation, and an inference caching mechanism. The objective of this architecture is to efficiently process both simple and complex natural language commands while ensuring semantic accuracy and logical consistency.
Formally, let an input query be represented as a token sequence x. The system predicts a set of intents $\hat{y}$ and corresponding slot values s, producing a structured output z. The overall system can be expressed as:

$$f(x) \rightarrow (y, s), \text{subject to} C(y, s) = \text{true}$$

where C denotes logical consistency constraints enforced by the symbolic knowledge graph.
The system follows a modular pipeline in which each component performs a specific role in the interpretation of user commands. Incoming queries are first normalized and evaluated for complexity. Let the normalized query be:

$$\tilde{x} = \text{Normalize}(x)$$

Based on this representation, a routing function R determines the processing path:

$$R(\tilde{x}) \in \{ML, DL\}$$

The predicted intents $\hat{y}$ are then enriched with slot extraction, producing structured slot values s, and validated through symbolic reasoning using a knowledge graph. The final structured command representation z is subsequently returned to the user and stored in a prediction cache D to optimize future queries.

3.1 System Architecture Overview

The proposed hybrid NLU framework combines neural prediction models with symbolic reasoning mechanisms to achieve accurate and efficient command interpretation. The overall architecture is illustrated in Figure 1.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

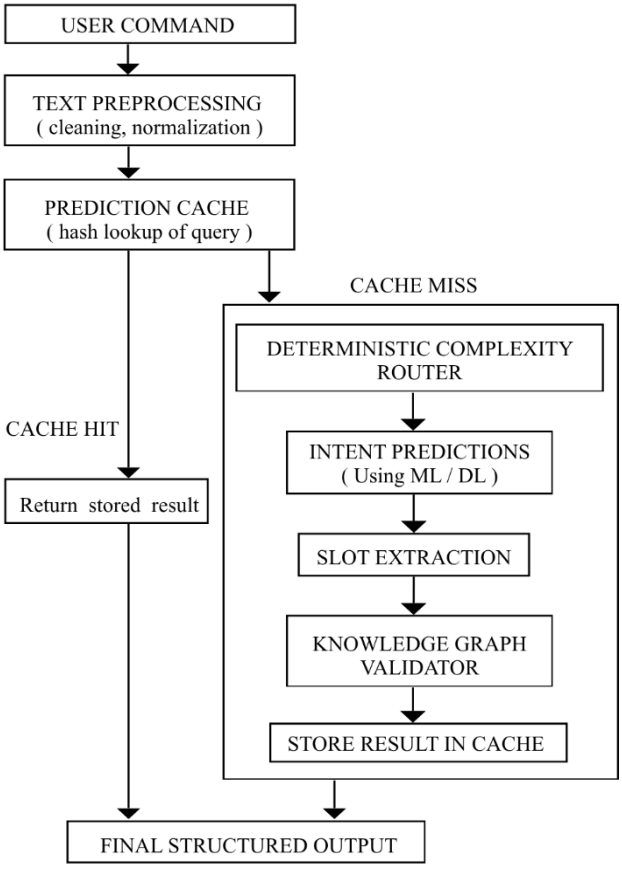


Figure 1: Proposed Hybrid Neural–Symbolic NLU System Architecture

The system is organized into several functional layers. The input processing layer prepares the raw user command for further processing through normalization and preprocessing operations. The infrastructure layer includes a prediction cache defined as:

$$D: h(x) \rightarrow (y, s)$$

where h is a hash function mapping normalized queries to stored predictions. This cache stores previously computed results to reduce redundant model inference.

The prediction layer contains both machine learning and deep learning models for intent detection. A deterministic routing mechanism determines which model should process a given query based on its structural complexity, formally expressed as:

$$R(x) = \begin{cases} \text{ML}, & \text{if query is simple} \\ \text{DL}, & \text{if query is complex} \end{cases}$$

Following intent prediction, a slot extraction module generates structured parameters required for command execution, producing slot values s. Finally, a knowledge graph validation layer verifies compatibility relationships between predicted intents $\hat{y}$ and extracted slots s to ensure logical consistency:

$$C(\hat{y}, s) \in \{\text{valid,invalid}\}$$

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

During runtime, a user query is first preprocessed and checked against the prediction cache. The cache lookup operation is defined as:

$$\text{if} h(x) \in D \Rightarrow (\hat{y}, s) = D[h(x)]$$

If a matching entry exists, the cached structured result is returned immediately. Otherwise, the router analyzes the query and directs it to either the machine learning classifier or the transformer-based deep learning model:

$$\hat{y} = \begin{cases} f_{ML}(x), & R(x) = ML \\ f_{DL}(x), & R(x) = DL \end{cases}$$

The predicted intents are then passed to the slot extraction module, which generates structured slot values using a generative language model:

$$s = g(x, \hat{y})$$

The resulting intent–slot combination is validated using the knowledge graph $C(\hat{y}, s)$, before being stored in the cache $D[h(x)] \leftarrow (\hat{y}, s)$ and returned as the final structured command.

This modular architecture enables the system to balance computational efficiency and semantic accuracy. Lightweight machine learning models handle simple queries with minimal latency, while transformer-based models provide deeper semantic understanding for complex or multi-intent commands.

3.2 Input Processing and Text Preprocessing

Before the query is processed by the prediction pipeline, the system performs text preprocessing to normalize the input and remove potential noise. Natural language commands often contain punctuation, irregular spacing, or inconsistent capitalization, which may affect downstream model performance if not handled appropriately.

Formally, let the raw input query be represented as a token sequence x. The preprocessing module applies a normalization function:

$$\tilde{x} = \text{Normalize}(x)$$

where $\tilde{x}$ denotes the normalized query.

The preprocessing module applies a sequence of normalization steps to transform the input into a consistent format. These steps include removing non-alphanumeric characters, converting text to lowercase, and collapsing multiple whitespace characters into a single space. This sequence of transformations can be expressed as a composition of functions:

$$\tilde{x} = f_{trim} \circ f_{space} \circ f_{lower} \circ f_{clean}(x)$$

where f_{clean} removes non-alphanumeric characters, f_{lower} converts text to lowercase, f_{space} normalizes whitespace, and f_{trim} removes leading and trailing spaces.

The normalized query is then trimmed to remove leading and trailing spaces. This preprocessing procedure ensures that semantically identical commands expressed with minor variations are treated consistently by the prediction models.

The preprocessing logic is implemented in the orchestration layer of the system, where the normalized query representation $\tilde{x}$ is generated before routing and prediction operations begin. By standardizing the input representation, the system improves both model reliability and cache effectiveness, since identical queries satisfy:

$$x_1 \sim x_2 \Rightarrow \text{Normalize}(x_1) = \text{Normalize}(x_2)$$

and can therefore be matched correctly during cache lookup.

3.3 Prediction Cache Mechanism

To reduce redundant computation and improve runtime efficiency, the system incorporates a prediction caching mechanism. The cache stores previously computed prediction results and allows the system to bypass model inference when the same query is encountered again.

Formally, the cache is defined as a mapping:

$$D: K \rightarrow (y, s)$$

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

where K is the set of hash keys derived from normalized queries, and (y, s) represents the predicted intent–slot pair.
The caching component is implemented using a hash-based lookup strategy. Each normalized query x is converted into a unique hash value using a cryptographic hashing function:

$$k=h(x)$$

The hash serves as the primary key for storing and retrieving prediction results from the cache. When a new query is received, the system first computes the corresponding hash and checks whether the cache already contains a stored result. The cache lookup operation is defined as:

$$\text{CacheHit}(x)=\begin{cases}\text{true}, & k \in D \\ \text{false}, & \text{otherwise}\end{cases}$$

If a cache hit occurs, the stored result is returned directly: $(y, s)=D[k]$. Otherwise, the system performs full inference and updates the cache: $D[k] \leftarrow (y, s)$.

The cache implementation maintains both an in-memory representation D_{mem} and a persistent database backend D_{db}. When the system starts, previously stored predictions are loaded from the database into memory for fast lookup:

$$D_{mem} \leftarrow D_{db}$$

During runtime, prediction results are written asynchronously to the database while the in-memory cache handles immediate retrieval operations. This hybrid design ensures both fast access and persistent storage of cached predictions.

To prevent unbounded cache growth, the system also implements an eviction strategy with a capacity constraint:

$$\| D_{mem} \| \leq C$$

where C is the maximum cache size. Entries with the lowest access frequency and oldest access timestamps are prioritized for removal. Let $f(k)$ denote access frequency and $t(k)$ denote last access time; eviction prioritizes entries minimizing:

$$\text{EvictScore}(k)=\alpha \cdot f(k)^{-1}+\beta \cdot t(k)$$

This policy ensures that frequently used predictions remain available while less relevant entries are gradually replaced. By eliminating repeated inference for identical queries, the caching mechanism significantly reduces system latency and improves scalability, particularly in scenarios where users frequently issue similar commands.

3.4 Deterministic Complexity Routing

The hybrid architecture relies on a deterministic routing mechanism to determine which prediction model should process a given query. Since user commands vary significantly in complexity, applying deep neural models to every query would introduce unnecessary computational overhead. The routing module addresses this issue by analyzing the structural characteristics of each query and directing it to the most appropriate prediction model.
Formally, given a normalized query x, the routing function is defined as:

$$R(x) \in \{ML, DL\}$$

The routing process begins by extracting structural features from the normalized query. These features are represented as a feature vector:

$$\phi(x)=\left[l(x), n_v(x), n_c(x), k(x), h_v(x)\right)$$

where $l(x)$ is token count, $n_v(x)$ is the number of detected action verbs, $n_c(x)$ is the number of independent clauses, $k(x)$ is a conjunction indicator (binary or count-based), and $h_v(x)$ is the diversity of action categories.

The router maintains a predefined vocabulary of action verbs V, partitioned into domain-specific subsets:

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

$$V = \bigcup_{d \in D} V_d$$

where V_d represents command domains such as file operations, system control, media operations, and network management. By detecting verbs and mapping them to domain categories, the system computes the diversity measure:

$$h_v(x) = |\{d | v \in V_d\})$$

which captures the heterogeneity of operations within the query.
Based on these extracted features, the router evaluates a set of deterministic rules that determine the routing decision. This rule-based decision function can be formalized as:

$$R(x) = \begin{cases} DL, & \text{if } n_c(x) \geq 2 \vee n_v(x) \geq 2 \vee h_v(x) \geq 2 \vee k(x) = 1 \vee l(x) > \tau_l \\ ML, & \text{otherwise} \end{cases}$$

where τ_l is a predefined length threshold.
Queries that contain multiple independent clauses, multiple action verbs, or verbs belonging to heterogeneous operation classes are classified as complex queries and are routed to the deep learning model. Similarly, long sentences containing conjunctions are also considered complex and handled by the transformer-based model. In contrast, queries that contain a single action verb and limited structural complexity are considered simple commands and are routed to the machine learning classifier.

This deterministic routing strategy provides several advantages. First, it significantly reduces computational cost by ensuring that simple queries are handled by lightweight models with faster inference times. Second, it allows the deep learning model to focus on more challenging queries that require deeper contextual reasoning. Finally, the rule-based routing design ensures predictable system behavior and avoids the training overhead associated with learned routing mechanisms.

3.5 Machine Learning-Based Intent Prediction Module
For simple commands with relatively predictable syntactic patterns, the system employs a lightweight machine learning classifier to perform intent detection. The machine learning layer is designed to provide fast inference while maintaining sufficient classification accuracy for structured commands.
The classification pipeline begins by converting the input query into a dense semantic representation using a sentence embedding model. Specifically, a pre-trained sentence transformer model is used to generate a vector representation of the input text:

$$e = Enc_{SBERT}(x) \in R^d$$

These embeddings capture semantic similarity between queries and provide a robust feature representation for downstream classification models.
The intent classification stage uses an ensemble of three supervised learning algorithms: Support Vector Machines (SVM), Logistic Regression, and Extreme Gradient Boosting (XGBoost). Each classifier independently predicts probability scores over the intent space I:

$$p^{(k)}(i|x) \in [0,1), i \in I, k \in \{1,2,3\}$$

The final prediction is obtained by combining these probabilities through a weighted aggregation scheme:

$$p_{ML}(i|x) = \sum_{k=1}^{3} \alpha_k \, p^{(k)}(i|x), \sum_{k=1}^{3} \alpha_k = 1$$

The weights α_k are determined during training based on validation performance and reflect the relative predictive strength of each classifier.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

The ensemble approach improves robustness by combining the strengths of multiple classifiers. Support Vector Machines provide strong performance for high-dimensional feature spaces, logistic regression offers stable probabilistic predictions, and gradient boosting models capture non-linear decision boundaries. By aggregating their predictions, the ensemble reduces the risk of overfitting and improves classification stability across different query patterns.
At inference time, the machine learning classifier returns the predicted intent:

$$\hat{y}=\arg\max_{i\in I} p_{ML}(i|x)$$

along with the corresponding confidence score $p_{ML}(\hat{y}|x)$. Since this model is primarily used for simple queries, the classification process remains computationally efficient and enables rapid system responses.

3.6 Transformer-Based Multi-Intent Prediction Module

For complex queries that involve multiple semantic actions or contextual dependencies, the system employs a transformer-based deep learning model for intent detection. Transformer architectures have demonstrated strong performance in natural language processing tasks due to their ability to model contextual relationships between tokens using self-attention mechanisms.
The deep learning component is built on a pre-trained transformer encoder architecture. Given the normalized input x, the encoder produces contextual embeddings:

$$H=\mathrm{Enc}_{\mathrm{Trf}}(x) \in R^{T\times d_h}$$

The representation of the special classification token is used as a global representation:

$$h=H_{CLS}$$

This representation is then passed through a shared feature encoding layer:

$$h'=\mathrm{LayerNorm}(W_s h+b_s)$$

Following the shared feature encoder, the model branches into two prediction heads. The first head performs multi-label intent classification:

$$p=\sigma(W_i h'+b_i), p \in [0,1)^{|I|}$$

The second head predicts the number of intents present in the query:

$$\hat{c}=\arg\max \mathrm{softmax}(W_c h'+b_c)$$

This dual-head architecture enables the model to capture both semantic intent categories and the structural complexity of the query. The final predicted intent set is determined using a hybrid selection strategy:

$$\hat{y}=\begin{cases}\text{Top-}\hat{c}\text{ intents by } p, & \text{if } P_c(\hat{c}) \geq \tau_c \\ \{i|p_i \geq \delta\}, & \text{otherwise}\end{cases}$$

where δ is the probability threshold, τ_c is the confidence threshold for count prediction, and $P_c(\hat{c})$ is the predicted confidence for intent count.
The model is trained using a joint loss function:

$$L=L_{BCE}(p,y)+\lambda L_{CE}(\hat{c},c)$$

where L_{BCE} is the binary cross-entropy loss for multi-label classification, L_{CE} is the cross-entropy loss for intent count prediction, and λ is a balancing parameter.

By combining contextual embeddings with a dual-head prediction architecture, the deep learning model provides robust semantic interpretation for complex user commands. Although transformer models require higher computational resources than traditional classifiers, the deterministic routing mechanism ensures that they are applied only when necessary, thereby maintaining overall system efficiency.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

3.7 Slot Extraction Module

After the intent prediction stage, the system performs slot extraction to identify structured parameters required for executing the predicted command. Slot extraction transforms natural language expressions into structured key–value representations that capture relevant entities such as file paths, application names, media types, and system parameters.

Formally, given the normalized query $\hat{x}$ and the predicted intent set $\hat{y}$, the slot extraction module predicts a structured slot representation:

$$s=g(\hat{x},\hat{y})$$

where s is a structured set of slot–value pairs:

$$s=\{(k_1,v_1),(k_2,v_2),\dots,(k_m,v_m)\}$$

The proposed system employs a sequence-to-structure generation model based on the T5 transformer architecture. Unlike traditional slot filling approaches that treat slot extraction as a token-level labeling problem, the sequence-to-structure formulation allows the model to generate structured slot representations directly from the input query. The generation process is modeled as a conditional sequence generation task:

$$P(s|\hat{x},\hat{y})=\prod_{t=1}^{L} P\left(s_t|s_{<t},\hat{x},\hat{y}\right)$$

where s_t represents the generated token sequence encoding the structured output.

During inference, the slot extraction model receives a composite input consisting of the user query and the predicted intent:

$$\hat{x}'=\text{Concat}(\hat{x},\hat{y})$$

This combined input allows the model to generate slot values that are consistent with the semantic context of the predicted command. The model generates a JSON-like representation containing the predicted slot values for the corresponding intent.

Following generation, the predicted slot values are processed through a post-processing stage that applies validation rules and constraints. Let the post-processing function be defined as:

$$s'=V(s,\hat{y})$$

where V enforces structural and semantic constraints, including enumerated slot value restrictions, conditional dependencies between slots, and intent–slot compatibility rules. Formally, validity is expressed as:

$$V(s,\hat{y})=\begin{cases}s', & \text{if satisfies all constraints}\\ \emptyset, & \text{otherwise}\end{cases}$$

For example, certain slot values are only valid when specific intent parameters are present, which can be expressed as:

$$(k,v)\in s \Rightarrow \text{Valid}(k,v|\hat{y})=\text{true}$$

The post-processing step ensures that the generated slot values conform to the predefined slot schema before being passed to the symbolic validation module. This generative slot extraction strategy provides several advantages over traditional sequence labeling methods by avoiding token-level ambiguity and producing outputs that can be immediately integrated into command execution pipelines.

3.8 Knowledge Graph Validation Layer

Although neural models are effective at predicting intents and extracting slots, they may occasionally produce predictions that violate domain constraints or contain logically inconsistent combinations of actions. To address this limitation, the proposed framework incorporates a symbolic reasoning layer based on a knowledge graph.

The knowledge graph is formally defined as a directed graph:

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

$$G=(V, E, A)$$

where V is the set of intent nodes, E is the set of edges representing relationships, and A is a set of node attributes (e.g., required slots, intent categories). Each intent $i \in V$ is associated with metadata:

$$\text{ReqSlots}(i) \subseteq K$$

where K is the set of all slot keys. The graph structure encodes two types of relations: conflict relations $E_{conflict}$ and dependency relations E_{depend}.

During validation, the predicted intent set $\hat{y}$ and extracted slot values s are evaluated against the rules defined in the knowledge graph. The validation function is defined as:

$$C(\hat{y}, s) \in \{\text{valid,invalid}\}$$

The validation process performs three checks. First, it verifies that all predicted intents belong to the valid intent inventory:

$$\hat{y} \subseteq V$$

Second, it checks whether the predicted combination of intents contains logical conflicts:

$$\exists (i, j) \in \hat{y} \times \hat{y} \text{such that} (i, j) \in E_{conflict} \Rightarrow \text{invalid}$$

For example, mutually exclusive commands such as simultaneous shutdown and restart operations are captured through conflict edges. Third, the system verifies that required slots for each intent are present and satisfy constraints:

$\forall i \in \hat{y}, \text{ReqSlots}(i) \subseteq \text{Keys}(s)$ and

$$(k, v) \in s \Rightarrow \text{Valid}(k, v|i)=\text{true}$$

where $\text{Keys}(s)$ denotes the set of slot keys in s.

The knowledge graph is implemented using a directed graph structure where edges represent relationships such as conflicts or dependencies between intents. This structure enables efficient reasoning about compatibility relationships during inference. By performing validation after neural inference, the symbolic layer ensures that the final structured command remains logically consistent with system rules. The integration of symbolic reasoning improves the reliability and interpretability of the NLU system by preventing invalid command interpretations and enforcing domain-specific constraints.

3.9 Hybrid Inference Pipeline

The hybrid NLU system integrates the previously described components into a unified inference pipeline that processes user commands from input to structured output. The pipeline is orchestrated by a central controller that coordinates the interaction between preprocessing, routing, prediction models, slot extraction, symbolic validation, and caching mechanisms.

Formally, the pipeline can be expressed as a sequence of transformations:

$$x \xrightarrow{\text{Normalize}} \tilde{x} \xrightarrow{\text{Cache}} \xrightarrow{R(\tilde{x})} \hat{y} \xrightarrow{g(\tilde{x},\hat{y})} s \xrightarrow{C(\hat{y},s)} z$$

When a user query is received, the system first applies text preprocessing to normalize the input representation: $\tilde{x}=\text{Normalize}(x)$. The normalized query is then checked against the prediction cache:

$$\text{if} h(\tilde{x}) \in D \Rightarrow (\hat{y}, s)=D\left[h(\tilde{x})\right)$$

If a matching cached entry is found, the stored structured result is returned immediately, eliminating the need for further computation.

If the query is not present in the cache, the deterministic router analyzes the query structure $R(\tilde{x}) \in \{\text{ML,DL}\}$. The selected model predicts the intent or set of intents:

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

$$\hat{y}=\begin{cases} f_{ML}(x), & R(x)=ML \\ f_{DL}(x), & R(x)=DL \end{cases}$$

These predicted intents are subsequently passed to the slot extraction module $s=g(x,\hat{y})$. The resulting intent–slot representation is then validated using the knowledge graph validator $C(\hat{y},s)$. After successful validation, the structured result is stored in the prediction cache $D[h(x)) \leftarrow (\hat{y},s)$, and the validated structured command representation z is returned as the system output.

This hybrid inference pipeline combines efficient model selection, deep semantic interpretation, and symbolic reasoning within a single modular architecture. By integrating these components, the system achieves a balance between computational efficiency, semantic accuracy, and logical reliability in command-driven natural language understanding.

IV. EXPERIMENTAL SETUP

This section describes the experimental configuration used to evaluate the proposed hybrid neural–symbolic natural language understanding framework. The experimental setup includes the dataset used for training and evaluation, the annotation schema adopted for representing intents and slots, and the statistical characteristics of the dataset. These elements establish the foundation for training the machine learning, deep learning, and slot extraction models described in Section 3 and provide the context for interpreting the experimental results presented in Section 5.

4.1 Dataset Description

The experiments were conducted using a command-oriented natural language dataset designed to simulate interactions with an intelligent offline assistant capable of executing system-level operations. The dataset consists of natural language queries representing common user commands such as file operations, application control, media playback, system monitoring, and device configuration. These commands were designed to reflect realistic interactions with command-driven interfaces used in desktop automation and intelligent assistant environments.

The dataset includes both **single-intent queries** and **multi-intent queries**. Single-intent queries represent commands that correspond to one system action, such as opening an application or searching for a file. Multi-intent queries contain multiple coordinated actions within a single sentence, for example commands that request several operations sequentially. Including both types of queries allows the dataset to evaluate the ability of the proposed framework to handle varying levels of linguistic complexity.

The dataset covers a predefined inventory of system operation intents that correspond to the command capabilities of the underlying assistant platform. Each query is associated with one or more intents from this inventory. In addition to intent labels, relevant slot values are annotated to capture the parameters required for executing the corresponding commands. These slots represent entities such as file names, application names, directories, device settings, and operational parameters.

The dataset was constructed to include diverse linguistic expressions for similar commands. Variations in phrasing, verb usage, and sentence structure were intentionally introduced to simulate natural user input and to improve the generalization capability of the trained models.

4.2 Dataset Statistics

A statistical analysis of the dataset was performed to understand the distribution of intents and slots across the collected queries. These statistics provide insight into the complexity and diversity of the dataset and help evaluate whether it adequately represents the range of commands handled by the system.

Table 1 presents an overview of the dataset, including the total number of samples, the number of distinct intents, and the number of slot types used in annotation. This table summarizes the overall scale and structural characteristics of the dataset.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

Table 2: Dataset Overview

Property	Value
Total Samples	15,720
Distinct Intents	20
Slot Types	19
Single-Intent Queries	7,085
Multi-Intent Queries	8,635

In addition to the overall dataset size, it is important to analyze how frequently different intents appear within the dataset. Table 2 presents the distribution of queries across different intent categories. This distribution illustrates how the dataset covers various system operations and ensures that the model is exposed to multiple examples of each command type during training.

Table 3: Intent Category Distribution

Category	Intents Included	# Intents	Total Samples	Percentage (%)
File Operations	create, rename, copy, move, delete, search	6	2,075	29.3
System Control	shutdown, restart, lock, virus scan	4	1,339	18.9
Application Control	open app, close app, running apps	3	980	13.8
Monitoring & Status	battery status, system usage	2	680	9.6
Media Control	Play media, pause media	2	770	10.8
Device Settings	adjust brightness, resize window, toggle connection	3	1,241	17.5
Total	—	20	7,085	100

Table 4 provides a detailed breakdown of the multi-intent command pairs present in the dataset, showing the variety of intent combinations and their frequencies.

Table 4: Multi-Intent Pair Distribution

ID	Intent Pair	Total Samples	Percentage (%)
1	adjust brightness + battery status	327	3.79
2	adjust brightness + open app	300	3.47
3	adjust brightness + play media	300	3.47
4	adjust brightness + resize window	286	3.31
5	adjust brightness + running apps	288	3.33
6	adjust brightness + system usage	300	3.47
7	battery status + running apps	198	2.29
8	battery status + toggle connection	173	2.00
9	close app + pause media	297	3.44
10	close app + resize window	300	3.47
11	close app + running apps	300	3.47
12	close app + shutdown system	300	3.47
13	copy file + move file	299	3.46
14	create file + delete file	300	3.47
15	delete file + file search	300	3.47
16	delete file + virus scan	284	3.29
17	file search + open app	300	3.47

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

ID	Intent Pair	Total Samples	Percentage (%)
18	file search + play media	299	3.46
19	lock system + shutdown system	294	3.40
20	move file + rename file	295	3.42
21	open app + play media	300	3.47
22	open app + resize window	299	3.46
23	pause media + play media	299	3.46
24	restart system + toggle connection	300	3.47
25	restart system + virus scan	293	3.39
26	running apps + shutdown system	300	3.47
27	running apps + system usage	251	2.91
28	shutdown system + toggle connection	277	3.21
29	shutdown system + virus scan	277	3.21
30	system usage + toggle connection	299	3.46
Total	—	8,635	100

4.3 Data Annotation Schema

To support intent detection and slot extraction tasks, the dataset was annotated using a structured schema that captures both the semantic action of a command and its associated parameters. Each query in the dataset is represented by three primary components: the raw command text, the associated intent label or set of intent labels, and the annotated slot values. Intent annotations identify the primary action or actions that the system should perform in response to the query. For single-intent queries, a single intent label is assigned to the command. For multi-intent queries, multiple intent labels are associated with the same query to represent coordinated system operations within a single sentence.

Slot annotations capture the parameters required to execute the predicted intents. Each slot corresponds to a specific entity type relevant to the system domain. Examples of slot types include application names, file names, directories, media content identifiers, and configuration parameters. Slot values are represented as key–value pairs where the key denotes the slot type and the value corresponds to the extracted entity from the query. Table 5 lists all intents and their associated slots used in the annotation.

Table 5: Intent-Slot Reference Table

S.No	Intent Name	Slot Name	Data Type
1	adjust_brightness	level	{string}
2	adjust_brightness	type	{enum: absolute | percentage | relative}
3	battery_status	info_type	{enum: health | percentage | time_remaining}
4	close_app	app	{string}
5	copy_file	source	{string}
6	copy_file	destination	{string}
7	create_file	file	{string}
8	delete_file	file	{string}
9	file_search	query	{string}
10	move_file	source	{string}
11	move_file	destination	{string}
12	open_app	app	{string}
13	play_media	item	{string}
14	rename_file	old_name	{string}
15	rename_file	new_name	{string}
16	resize_window	window	{string}
17	resize_window	resize_type	{enum: fullscreen | maximize | minimize | restore | set | snap}
18	resize_window	size	{string}
19	resize_window	position	{enum: top | bottom | left | right | top-left | top-right | bottom-left | bottom-right}
20	running_apps	filter	{string}
21	system_usage	resource	{enum: cpu | disk | memory}
22	toggle_connection	connection	{enum: bluetooth | wifi}
23	toggle_connection	action	{enum: on | off | toggle}
24	virus_scan	target	{string}

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

4.4 Model Training Configuration

The hybrid NLU framework integrates three independently trained components: a machine learning intent classifier, a transformer-based deep learning intent detection model, and a sequence-to-structure slot extraction model. Each component was trained separately using dedicated training scripts and later integrated into the unified inference pipeline described in Section 3.

For the machine learning component, sentence embeddings were first generated from the input queries using a pre-trained sentence transformer model. These embeddings served as feature vectors for supervised classification models. Three classifiers were trained: Support Vector Machine (SVM), Logistic Regression, and Extreme Gradient Boosting (XGBoost). During training, each model learned to predict the intent labels associated with the queries. The final machine learning classifier used during inference is an ensemble of these models, where prediction probabilities from each classifier are combined using a weighted aggregation scheme. This ensemble design improves robustness and reduces model bias compared with relying on a single classifier.

The deep learning component for intent detection was trained using a transformer-based encoder architecture. A pre-trained language model was fine-tuned on the command dataset to learn contextual semantic representations of user queries. The model architecture includes a shared encoder followed by two prediction heads: one for multi-label intent classification and another for predicting the number of intents present in the query. During training, the model optimized a joint loss function that combines intent classification loss and intent-count prediction loss. Fine-tuning was performed using stochastic gradient-based optimization with mini-batch training.

For slot extraction, a generative language model based on the T5 architecture was trained to convert natural language queries into structured slot representations. The training data consisted of pairs of input queries and corresponding structured slot outputs. The model was trained to generate slot values in a structured format that represents the parameters required for executing the predicted command. During training, teacher forcing was used to guide sequence generation, enabling the model to learn accurate mapping between natural language expressions and slot structures.

Each model was trained independently and evaluated on a validation subset of the dataset to select the best-performing configuration before integration into the hybrid system. This modular training approach allows individual components to be improved or replaced without modifying the overall system architecture.

4.5 Evaluation Metrics

To evaluate the performance of the proposed hybrid NLU framework, standard evaluation metrics commonly used in natural language understanding research were employed. These metrics measure the effectiveness of the system in both intent detection and slot extraction tasks.

Intent Detection Metrics

For intent detection, the evaluation focuses on classification accuracy as well as precision, recall, and F1-score. Since the system supports multi-intent queries, evaluation metrics are computed using a multi-label classification framework. This approach accounts for scenarios where multiple intents are associated with a single query and ensures that predictions are evaluated against all relevant intent labels.

Let N denote the total number of test samples, I denote the total number of distinct intent classes, $y_{ij} \in \{0,1\}$ indicate the ground truth presence of intent j in sample i, and $\hat{y}_{ij} \in \{0,1\}$ indicate the predicted presence of intent j in sample i. The evaluation metrics are defined as follows:

Accuracy measures the proportion of correctly predicted intent labels across all samples and all intent classes. For multi-label classification, accuracy is defined as the ratio of correctly predicted labels to the total number of labels:

$$\text{Accuracy} = \frac{1}{N \cdot I} \sum_{i=1}^{N} \sum_{j=1}^{I} 1\left(y_{ij} = \hat{y}_{ij}\right)$$

where $1(\cdot)$ is the indicator function that returns 1 if the condition is true and 0 otherwise.

Precision evaluates the proportion of predicted intents that are correct. For multi-label classification, macro-averaged precision is computed as:

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

$$\text{Precision}=\frac{1}{I}\sum_{j=1}^{I}\frac{TP_j}{TP_j+FP_j}$$

where TP_j (true positives) is the number of samples where intent j was correctly predicted, and FP_j (false positives) is the number of samples where intent j was incorrectly predicted.

Recall measures the proportion of actual intents that are correctly identified by the model. Macro-averaged recall is computed as:

$$\text{Recall}=\frac{1}{I}\sum_{j=1}^{I}\frac{TP_j}{TP_j+FN_j}$$

where FN_j (false negatives) is the number of samples where intent j was present in the ground truth but not predicted.

F1-Score represents the harmonic mean of precision and recall, providing a balanced measure of classification performance:

$$\text{F1-Score}=2\times\frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$$

Alternatively, the macro-averaged F1-score across all intent classes can be expressed as:

$$\text{F1-Score}=\frac{1}{I}\sum_{j=1}^{I}2\times\frac{\text{Precision}_j\times\text{Recall}_j}{\text{Precision}_j+\text{Recall}_j}$$

Slot Extraction Metrics

For slot extraction, slot accuracy is used as the primary evaluation metric. Let S denote the total number of slot instances in the dataset, where each slot instance is a key–value pair (k, v) associated with a query. Let s_{true} represent the ground truth slot set and s_{pred} represent the predicted slot set for a given query.

Slot Accuracy measures the proportion of correctly generated slot values relative to the total number of slots present in the dataset:

$$\text{Slot Accuracy}=\frac{1}{S}\sum_{i=1}^{N}\sum_{(k,v)\in s_{true}^{(i)}}1\left((k,v)\in s_{pred}^{(i)}\right)$$

where $s_{true}^{(i)}$ and $s_{pred}^{(i)}$ denote the ground truth and predicted slot sets for sample i, respectively.

Slot Precision evaluates the proportion of predicted slot values that are correct:

$$\text{Slot Precision}=\frac{\sum_{i=1}^{N}\left|s_{true}^{(i)}\cap s_{pred}^{(i)}\right)}{\sum_{i=1}^{N}\left|s_{pred}^{(i)}\right)}$$

Slot Recall measures the proportion of ground truth slot values that are correctly identified:

$$\text{Slot Recall}=\frac{\sum_{i=1}^{N}\left|s_{true}^{(i)}\cap s_{pred}^{(i)}\right)}{\sum_{i=1}^{N}\left|s_{true}^{(i)}\right)}$$

Slot F1-Score provides a balanced measure of slot extraction performance:

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

$$\text{Slot F1-Score} = 2 \times \frac{\text{Slot Precision} \times \text{Slot Recall}}{\text{Slot Precision} + \text{Slot Recall}}$$

Latency Metrics

Latency is also considered as an operational performance metric, since the hybrid architecture aims to balance accuracy with computational efficiency. The average end-to-end inference time is measured across queries to evaluate the impact of the deterministic routing mechanism and caching strategy on system responsiveness.

Let t_i denote the total processing time for query i from input normalization to structured output generation. The average latency is defined as:

$$\text{Avg. Latency} = \frac{1}{N}\sum_{i=1}^{N} t_i$$

where N is the total number of queries processed during evaluation. This metric captures the overall responsiveness of the system and enables comparison across different architectural configurations.

4.6 Hardware and Software Environment

All experiments were conducted in a controlled computational environment using widely adopted machine learning and deep learning frameworks. The models were implemented using the Python programming language and trained using open-source libraries commonly used for natural language processing research.

Hardware Configuration

The experiments were conducted on a computer system with the following hardware configuration:

- **Processor**: AMD Ryzen 5 5625U
- **RAM**: 4 GB
- **Storage**: Solid State Drive (SSD)
- **GPU**: Not required for inference (optional for training)
- **Operating System**: Ubuntu 22.04.5 (Linux)

The system was executed on a workstation equipped with a multi-core processor and sufficient memory resources to support both machine learning and deep learning experiments. GPU acceleration was used during the training of transformer-based models to improve training efficiency, while inference experiments were conducted in an environment representative of real-world deployment conditions. Since the system is designed for offline intelligent agents operating on standard hardware, GPU is not required for inference, making the architecture suitable for deployment on resource-constrained devices.

Software Environment

The software environment used for development and experimentation is as follows:

- **Programming Language**: Python 3.x
- **Development Environment**: Visual Studio Code / Jupyter Notebook

The deep learning components, including the transformer-based intent detection model and the T5 slot extraction model, were implemented using the PyTorch framework and the Hugging Face Transformers library. These frameworks provide optimized implementations of transformer architectures and efficient tools for model fine-tuning and inference.

The machine learning classifiers used for intent detection were implemented using the scikit-learn library, while the gradient boosting classifier was implemented using the XGBoost framework. Sentence embeddings used for feature generation were produced using a pre-trained sentence transformer model.

All experiments were conducted using a consistent software environment to ensure reproducibility of results. The modular design of the hybrid NLU system allows the trained models to be deployed across different computational environments without significant modification to the architecture.

Key software libraries and their versions used in the experiments are summarized in Table 3.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

Table 6: Software Library Versions

Library	Version
Python	3.10+
PyTorch	2.0+
Transformers (Hugging Face)	4.30+
scikit-learn	1.2+
XGBoost	1.7+
sentence-transformers	2.2+
NumPy	1.24+
Pandas	1.5+

V. RESULTS

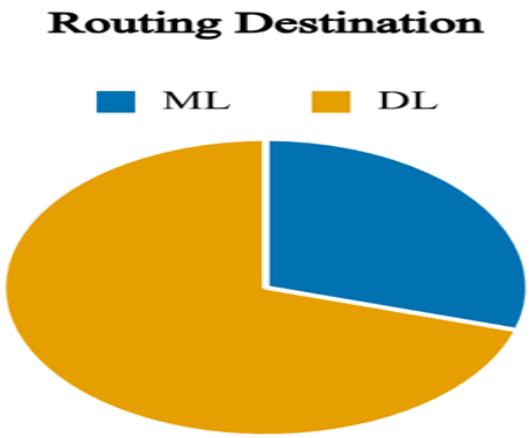
This section presents the experimental evaluation of the proposed hybrid neural–symbolic natural language understanding (NLU) framework. The evaluation focuses on five aspects of system performance: the effectiveness of the deterministic routing mechanism, machine learning–based intent classification, deep learning–based intent detection, slot extraction performance, and the overall hybrid architecture efficiency. All experiments were conducted under the experimental conditions described in Section 4. The results are reported using standard evaluation metrics widely used in NLU research, including accuracy, precision, recall, and F1-score for intent detection and slot extraction tasks.

5.1 Routing Performance by Query Complexity

The first experiment evaluates the effectiveness of the deterministic routing mechanism responsible for directing queries to either the machine learning (ML) classifier or the deep learning (DL) model based on estimated query complexity.

The routing performance is summarized in Table 4 and visualized in Figure 2.
Figure 2: Routing distribution by query complexity

Table 7: Routing Performance by Query Complexity



Query Complexity	Routed to ML	Routed to DL	Total Queries	Recall (%)	Precision (%)
Low Complexity	4,619	2,456	7,085	65.45	99.12
High Complexity	41	8,594	8,635	99.53	77.77
Overall	4,660	11,050	15,720	84.17	87.39

According to Table 4, the evaluation dataset contains 15,720 total queries, consisting of 7,085 low-complexity queries and 8,635 high-complexity queries. Among the low-complexity queries, 4,619 queries were routed to the ML classifier, while 2,456 queries were routed to the DL model, resulting in a recall of 65.45% and an exceptionally high precision of

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

99.12% for low-complexity detection. This indicates that the routing mechanism is highly reliable in identifying queries that can be efficiently handled by lightweight ML models.

For high-complexity queries, the router directed 8,594 queries to the DL model, while only 41 queries were incorrectly routed to the ML model. This corresponds to a recall of 99.53% and a precision of 77.77% for high-complexity routing. These results demonstrate that the routing mechanism is highly effective in ensuring that complex semantic queries are processed by the more expressive deep learning component.

Overall, the routing system directed 4,660 queries to the ML model and 11,050 queries to the DL model, achieving an overall recall of 84.17% and an overall precision of 87.39%. These results indicate that deterministic routing successfully partitions the workload between lightweight and deep models, enabling efficient processing without sacrificing semantic accuracy.

Routing mechanisms have recently emerged as an important strategy for improving scalability in natural language processing architectures. For example, mixture-of-experts models dynamically route inputs to specialized components to improve efficiency and model capacity. In comparison, the deterministic routing strategy used in this work provides a simpler and computationally lightweight alternative suitable for offline intelligent agents and resource-constrained systems.

5.2 Machine Learning Intent Classification Performance

The second experiment evaluates the performance of several machine learning algorithms for intent classification. The evaluated classifiers include Support Vector Machine (SVM), Random Forest, XGBoost, and a combined ensemble classifier.

The results are reported in Table 5 and illustrated in Figure 3.

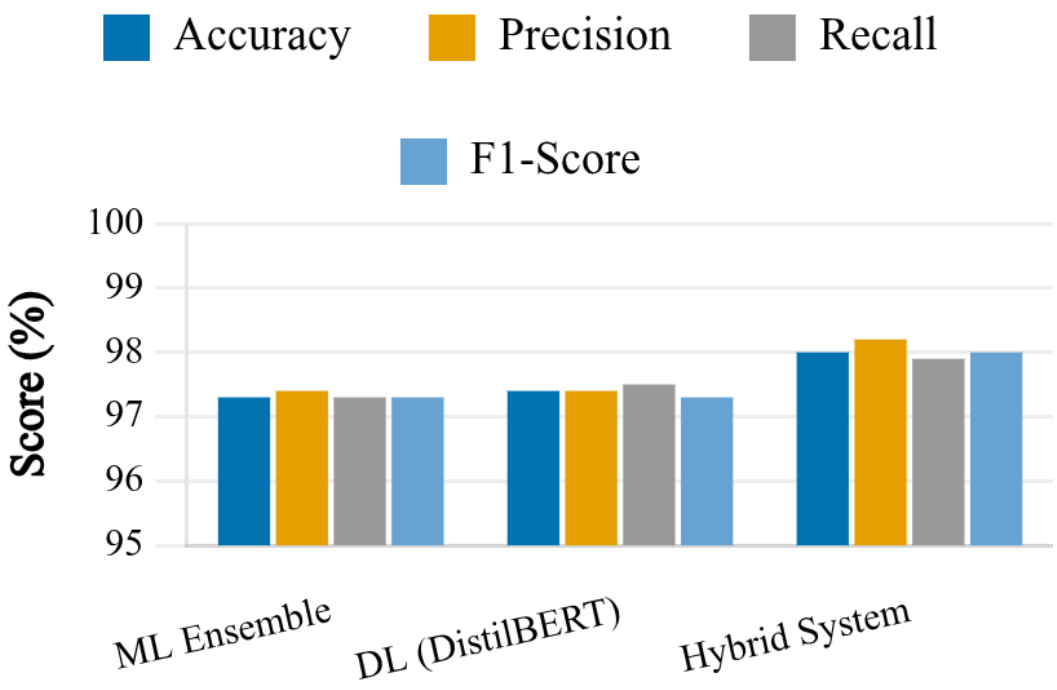


Figure 3: Comparison of intent classification performance across ML models

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

Table 8: ML Intent Classification Performance

Model	Train Acc. (%)	Test Acc. (%)	Precision (%)	Recall (%)	F1-Score (%)
SVM	99.7	96.9	97.1	97.1	96.9
Random Forest	99.0	96.9	97.0	97.1	96.8
XGBoost	99.8	95.0	95.0	95.0	94.8
Ensemble	99.8	97.3	97.4	97.3	97.3

Among the evaluated models, the ensemble classifier achieved the best overall performance. Specifically, the ensemble model achieved a training accuracy of 99.8% and a test accuracy of 97.3%, with a precision of 97.4%, recall of 97.3%, and an F1-score of 97.3%. These results demonstrate that combining multiple classifiers improves generalization and stability compared with individual models.

In comparison, both SVM and Random Forest achieved 96.9% test accuracy, with F1-scores of 96.9% and 96.8%, respectively. XGBoost achieved a slightly lower test accuracy of 95.0%, with an F1-score of 94.8%. Although these models perform well individually, the ensemble classifier provides a more balanced trade-off between precision and recall.

Traditional machine learning models remain competitive for intent detection tasks when queries are relatively simple or follow consistent syntactic patterns. Previous studies on intent classification have demonstrated that models such as SVM and Random Forest can achieve strong performance when appropriate feature representations are available. However, these models often struggle with complex multi-intent queries that require deeper contextual understanding.

5.3 Deep Learning Intent Detection Performance

To evaluate the effectiveness of deep neural architectures for intent detection, experiments were conducted using the distilbert-base transformer model under two different training configurations: a single-intent dataset and a double-intent dataset.
The results are presented in Table 6.

Table 9: DL Intent Detection Performance

Model	Data Type	Train Acc. (%)	Test Acc. (%)	Precision (%)	Recall (%)	F1-Score (%)
distilbert-base	Single	98.7	96.5	96.6	96.8	96.5
distilbert-base	Double	98.5	97.6	98.1	98.1	98.1
distilbert-base	Overall	98.6	97.4	97.4	97.5	97.3

For the single-intent dataset, the model achieved a training accuracy of 98.7% and a test accuracy of 96.5%, with precision and recall values of 96.6% and 96.8%, respectively. The resulting F1-score was 96.5%. When trained on the double-intent dataset, the model demonstrated improved performance with a test accuracy of 97.6%, while both precision and recall reached 98.1%, producing an F1-score of 98.1%.

These results indicate that transformer-based models are particularly effective at capturing semantic relationships in complex multi-intent queries. Unlike traditional machine learning classifiers, deep neural architectures leverage contextual embeddings that capture long-range dependencies between tokens, enabling more accurate interpretation of complex natural language commands.

Recent advances in transformer-based language models have significantly improved performance across various NLU tasks, including intent detection and slot filling. The results observed in this experiment confirm that transformer models provide strong semantic representation capabilities, particularly for multi-intent command understanding.

5.4 Slot Extraction Performance

The fourth experiment evaluates the effectiveness of the slot extraction module responsible for identifying and structuring semantic entities from user utterances. In the proposed architecture, slot extraction is performed using a T5-small sequence-to-structure model, which converts natural language input into structured parameter representations required for command execution.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

The slot extraction performance is presented in Table 7 and visualized in Figure 4.

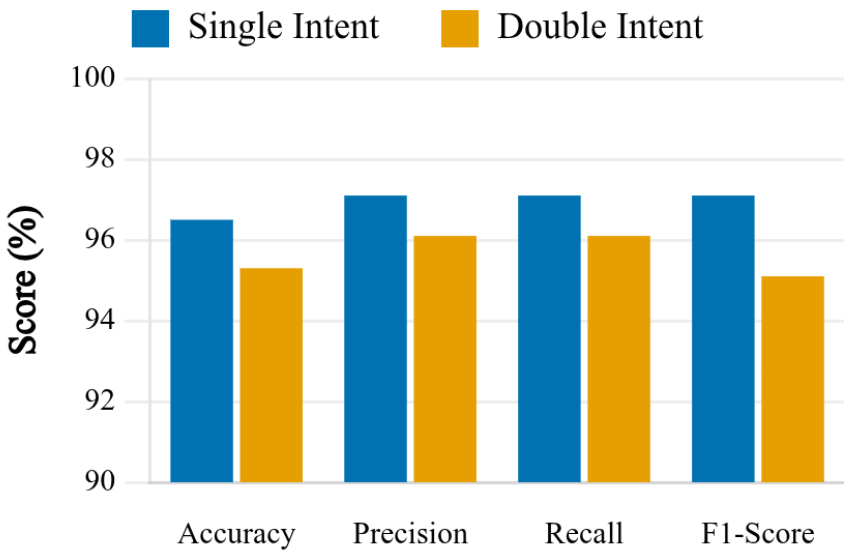


Figure 4: Slot extraction performance across dataset types

Table 10: Slot Extraction Performance

Method	Data Type	Train Acc. (%)	Test Acc. (%)	Precision (%)	Recall (%)	F1-Score (%)
t5-small	Single	99.8	96.5	97.1	97.1	97.1
t5-small	Double	99.3	95.3	96.1	96.1	95.1
t5-small	Overall	99.5	95.9	96.6	96.6	96.1

The results show that the slot extraction module achieves high accuracy across both single-intent and multi-intent scenarios. For the single-intent dataset, the model achieved a training accuracy of 99.8% and a test accuracy of 96.5%, with precision, recall, and F1-score values all reaching 97.1%. These results indicate that the model is capable of accurately identifying semantic entities within relatively simple user queries while maintaining strong generalization performance.

For the double-intent dataset, which contains more complex commands involving multiple semantic targets, the slot extraction model achieved a training accuracy of 99.3% and a test accuracy of 95.3%, with precision and recall values of 96.1% and an F1-score of 95.1%. Although performance slightly decreases compared with the single-intent dataset, the model still maintains a high level of accuracy despite the increased complexity of the input queries.

These results demonstrate that the sequence-to-structure formulation is effective for semantic slot extraction tasks. Unlike traditional sequence labeling methods that assign labels to each token independently, the T5-based approach generates structured outputs directly from the input utterance. This design enables the model to capture global contextual relationships within the sentence, which is particularly important when multiple slots are associated with multiple intents.

Previous studies on joint NLU architectures have demonstrated the benefits of hierarchical modeling for slot extraction. For example, Zhang et al. [3] showed that capsule-based neural networks can capture hierarchical relationships between words, slots, and intents. Similarly, transformer-based models have been widely adopted for slot filling due to their ability to model long-range contextual dependencies. The results observed in this experiment confirm that sequence-to-structure transformer models provide strong performance for slot extraction while also generating outputs that can be directly integrated into command execution pipelines.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

5.5 Hybrid System Performance

The fifth experiment evaluates the overall performance of the proposed hybrid NLU architecture by comparing it with two alternative system configurations: a machine learning pipeline combined with slot extraction (ML + Slot) and a deep learning pipeline combined with slot extraction (DL + Slot). This comparison allows the effectiveness of the hybrid routing strategy to be evaluated in terms of both prediction accuracy and system latency. The comparative results are presented in Table 8 and visualized in Figure 5.

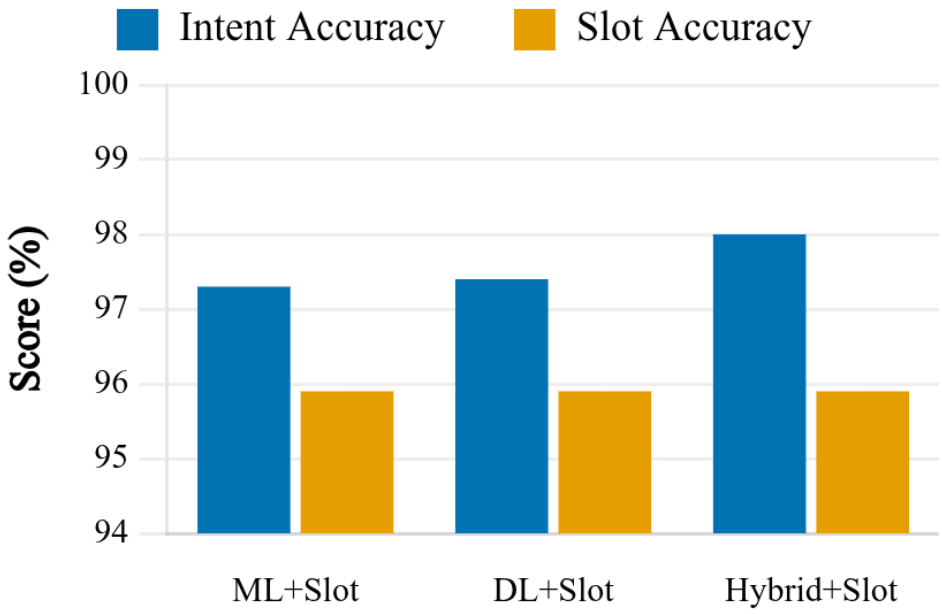


Figure 5: Performance comparison of hybrid system vs. individual architectures

Table 11: Hybrid System vs. Individual Architectures

System	Intent Accuracy (%)	Slot Accuracy (%)	Avg. Latency (ms)
ML + Slot	97.3	95.9	316.7
DL + Slot	97.4	95.9	363.8
Hybrid + Slot	98.0	95.9	333.4

The ML + Slot configuration achieved an intent accuracy of 97.3% and a slot accuracy of 95.9%, with an average latency of 316.7 ms. While this configuration provides relatively efficient inference, its performance is limited when processing complex or multi-intent queries due to the restricted expressive capacity of traditional machine learning classifiers.

The DL + Slot configuration improved intent classification performance, achieving an intent accuracy of 97.4%. However, slot extraction performance remained at 95.9%, and the average latency increased to 363.8 ms. This increase in latency is primarily due to the computational overhead associated with transformer-based models.

In contrast, the proposed Hybrid + Slot architecture achieved the best overall performance, with an intent accuracy of 98.0%, slot accuracy of 95.9%, and an average latency of 333.4 ms. The hybrid system therefore improves intent detection accuracy by 0.7% compared with the ML pipeline and 0.6% compared with the DL pipeline, while maintaining comparable slot accuracy.

These results demonstrate that the deterministic routing mechanism effectively allocates queries to the most appropriate prediction model. Simple queries can be processed efficiently using lightweight ML classifiers, while more complex

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

multi-intent queries are directed to the transformer-based DL model. By combining the strengths of both approaches, the hybrid system achieves superior accuracy while maintaining competitive inference latency.

Hybrid architectures have recently gained attention in natural language processing as a strategy for balancing computational efficiency and semantic modeling capability. Similar ideas have been explored in mixture-of-experts frameworks, where inputs are dynamically routed to specialized components. However, the deterministic routing mechanism used in this work provides a simpler and computationally lightweight alternative that is particularly suitable for offline intelligent agents.

5.6 Latency Evaluation

The final experiment evaluates the computational efficiency of the proposed NLU pipeline by measuring the average processing time of each system component. Latency is an important factor for real-time conversational systems, where slow response times can significantly degrade user experience.

The latency measurements for each component are presented in Table 9 and visualized in Figure 6 and Figure 7.

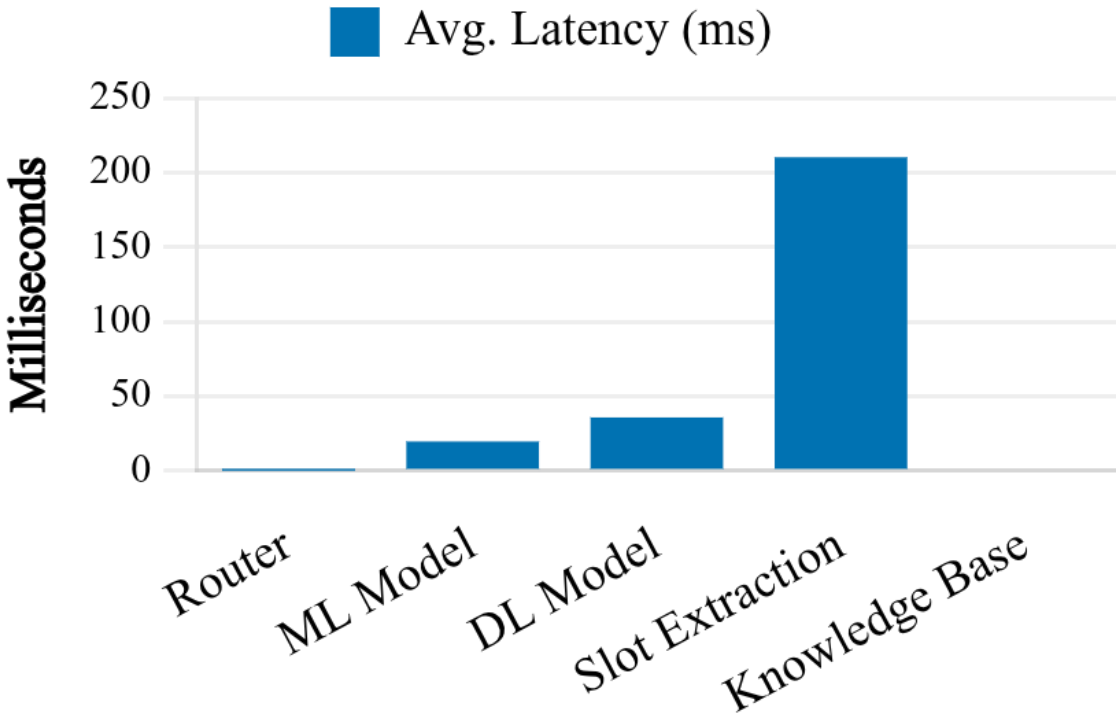


Figure 6: Latency breakdown by system component

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

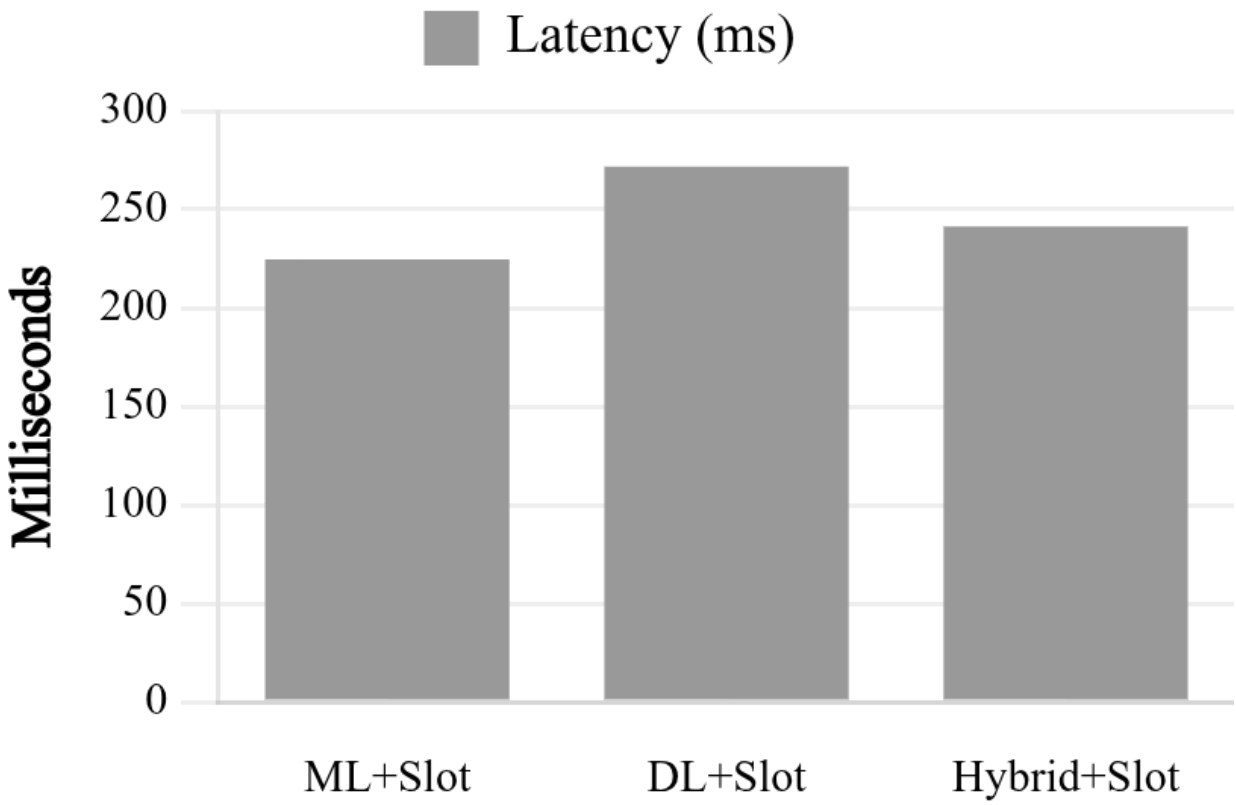


Figure 7: Comparison of end-to-end latency across system variants

Table 12: Latency Evaluation by Component

Component	Average Latency (ms)
Router	0.1
ML Model	19.7
DL Model	35.8
Slot Extraction	302.3
Knowledge Base	0.0
End-to-End Pipeline	333.4

The results show that the deterministic router introduces only 0.1 ms of latency, making it effectively negligible in the overall processing pipeline. The machine learning classifier requires an average of 19.7 ms per query, while the deep learning model requires 35.8 ms due to the computational complexity of transformer-based inference.

The slot extraction module represents the most computationally intensive component, requiring an average processing time of 302.3 ms. This is expected because sequence-to-structure generation models perform autoregressive decoding, which involves multiple token generation steps.

The knowledge graph validation layer introduces negligible computational overhead, as compatibility checks are implemented through lightweight rule-based operations.

Overall, the complete NLU pipeline achieves an end-to-end latency of 333.4 ms, which demonstrates that the hybrid architecture can maintain high semantic accuracy while still meeting the efficiency requirements of real-time conversational systems.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

These results confirm that the deterministic routing mechanism significantly contributes to improving computational efficiency by reducing the number of queries processed by the deep learning model. Instead of applying expensive transformer inference to all queries, the system selectively invokes deep learning only when necessary. This strategy enables the system to achieve both high accuracy and efficient inference, making it suitable for deployment in practical command-driven intelligent agents.

VI. DISCUSSION

The results presented in the previous section show that the proposed hybrid neural–symbolic NLU framework improves both semantic accuracy and computational efficiency. Beyond numerical gains, these findings are analyzed in relation to existing work in hybrid architectures, routing strategies, transformer-based modeling, slot extraction, and neural–symbolic reasoning.

6.1 Interpretation of Hybrid Architecture Performance

The hybrid architecture outperforms standalone ML and DL systems, achieving 98.0% intent accuracy and 95.9% slot accuracy with an end-to-end latency of 333.4 ms. This indicates that combining modeling paradigms yields more balanced performance than relying on a single approach.

Traditional ML models perform well on structured queries but struggle with complex semantics, while transformer models capture contextual dependencies at higher computational cost. The proposed architecture leverages both strengths by assigning each model to suitable query types.

Unlike prior work such as CNN–BiLSTM models or graph-based approaches like AGIF, which focus on improving single-model representations, the proposed system emphasizes **system-level design**. The results show that architectural decisions, not just model complexity, are critical for high-performance NLU.

6.2 Effectiveness of Deterministic Routing

The deterministic routing mechanism enables efficient workload distribution by directing simple queries to ML models and complex queries to DL models. This reduces unnecessary computation while preserving semantic accuracy.
Compared to mixture-of-experts models that rely on learned gating, the proposed rule-based routing is simpler and avoids training instability. The results show effective separation of query complexity, contributing directly to latency reduction.

This demonstrates that deterministic routing is a practical alternative for efficient NLU in resource-constrained environments.

6.3 Role of Transformer-Based Semantic Modeling

The transformer-based model performs strongly on complex and multi-intent queries, achieving high accuracy and recall. Its self-attention mechanism enables effective modeling of contextual dependencies across tokens.
While prior works such as AGIF and MISCA enhance representation learning within deep models, they apply the same architecture to all queries. In contrast, the proposed system uses transformers selectively, preserving their strengths while reducing computational overhead.
This confirms that transformer models are most effective when integrated within a hybrid framework rather than used in isolation.

6.4 Effectiveness of Sequence-to-Structure Slot Extraction

The T5-based slot extraction model achieves strong performance across both single and multi-intent datasets. Unlike traditional sequence labeling methods, this approach generates structured outputs directly from the input sequence.
Earlier methods such as CRF-based models or capsule networks model token-level or hierarchical relationships but may miss global context. The sequence-to-structure approach captures sentence-level semantics and produces execution-ready outputs.
These results indicate that generative slot extraction is a suitable alternative for command-oriented NLU systems.

6.5 Contribution of Neural–Symbolic Integration

The knowledge graph validation layer ensures logical consistency by verifying predicted intents and slot values against domain constraints. This addresses limitations of neural models, which may produce semantically valid but logically inconsistent outputs.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

Unlike approaches that embed symbolic knowledge within neural models, the proposed system uses post-inference validation. This separation allows neural models to focus on prediction while the symbolic layer enforces constraints.
The results demonstrate that this design improves reliability without increasing model complexity.

6.6 Implications for Real-World Intelligent Agents
The proposed framework provides a practical balance between accuracy and efficiency for real-world systems. By combining ML and DL models with routing, the system achieves low latency suitable for interactive applications.
The integration of symbolic validation further enhances reliability, reducing the risk of incorrect or unsafe actions in command-driven environments.

Overall, the findings suggest that future NLU systems can benefit from **hybrid, modular architectures** that integrate complementary techniques rather than relying solely on increasingly complex neural models.

VII. FUTURE WORK

Although the proposed hybrid neural–symbolic NLU framework demonstrates strong performance in both accuracy and efficiency, several directions remain for further improvement and extension.

7.1 Adaptive and Learning-Based Routing
The current deterministic routing effectively separates simple and complex queries but relies on fixed rules. Future work can explore adaptive routing strategies that learn decisions from data. Approaches such as gating networks, reinforcement learning, or mixture-of-experts frameworks could enable dynamic and optimized workload distribution, further improving accuracy and latency.

7.2 Dynamic Knowledge Graph Expansion
The knowledge graph currently depends on manually defined rules. Future research can focus on automatic expansion using knowledge extraction from text corpora, logs, or documentation. Techniques such as ontology learning and probabilistic reasoning could enhance validation capabilities and allow the symbolic layer to evolve with the system.

7.3 Advanced Multi-Intent Modeling
While the system performs well for single and dual intents, real-world queries often involve more complex and hierarchical commands. Future extensions can incorporate hierarchical intent representations and task decomposition methods to handle multi-step and dependent operations more effectively.

7.4 Cross-Domain and Large-Scale Evaluation
The current evaluation is limited to a command-driven dataset. Future work should test the framework on larger, cross-domain datasets such as conversational systems and task-oriented dialogues. Multilingual evaluation can also be explored to assess generalization across languages.

7.5 Multimodal Integration
The framework currently processes textual input only. Future research can integrate speech, visual context, and sensor data to enable multimodal understanding. This would improve contextual awareness and support more realistic intelligent assistant applications.
In summary, future work can enhance the framework through adaptive routing, dynamic knowledge graphs, advanced intent modeling, broader evaluation, and multimodal integration.

VIII. CONCLUSION

This work proposed a hybrid neural–symbolic framework for multi-intent natural language understanding that balances semantic accuracy, efficiency, and logical consistency. Unlike traditional systems that rely on a single modeling paradigm, the proposed architecture integrates deterministic routing, machine learning classifiers, transformer-based deep learning, sequence-to-structure slot extraction, and knowledge graph validation.

The routing mechanism directs simple queries to efficient ML models while assigning complex queries to DL models, enabling effective use of computational resources. Experimental results show that the system achieves 98.0% intent accuracy and 95.9% slot accuracy with an average latency of approximately 333 ms, demonstrating improvements over standalone approaches.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

The integration of a knowledge graph further enhances reliability by ensuring logical consistency in predicted outputs. This neural–symbolic design allows neural models to focus on semantic prediction while symbolic validation enforces domain constraints.

Importantly, the proposed framework represents a modular architectural design rather than a fixed implementation. Individual components, including routing strategies, model architectures, slot extraction methods, and knowledge graph structures, can be modified based on application requirements and datasets.

Overall, the results demonstrate that combining machine learning, deep learning, and symbolic reasoning within a unified architecture provides an effective and scalable solution for multi-intent NLU. This work highlights the importance of system-level design and offers a flexible foundation for future intelligent agent systems.

REFERENCES

1. A. S. M. Zailan, N. H. I. Teo, N. A. S. Abdullah, and M. Joy, "State of the Art in Intent Detection and Slot Filling for Question Answering System: A Systematic Literature Review," International Journal of Advanced Computer Science and Applications, vol. 14, no. 11, pp. 15–27, 2023.
2. D. Wu, R. Fang, L. Jiang, S. Song, X. Huang, S. Wang, Z. Li, L. Shi, M. Bao, Y. Li, and H. Huang, "Multi-Intent Spoken Language Understanding: A Survey of Methods, Trends, and Challenges," Vicinagearth, vol. 2, 2025.
3. C. Zhang, Y. Li, N. Du, W. Fan, and P. S. Yu, "Joint Slot Filling and Intent Detection via Capsule Neural Networks," Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), 2019.
4. Y. I. Muhammad, N. Salim, and A. Zainal, "Joint Intent Detection and Slot Filling with Syntactic and Semantic Features Using Multichannel CNN-BiLSTM," PeerJ Computer Science, 2024.
5. L. Qin, X. Xu, W. Che, and T. Liu, "AGIF: An Adaptive Graph-Interactive Framework for Joint Multiple Intent Detection and Slot Filling," Findings of the Association for Computational Linguistics: EMNLP, 2020.
6. Z. Ding, Z. Yang, H. Lin, and J. Wang, "Focus on Interaction: A Novel Dynamic Graph Model for Joint Multiple Intent Detection and Slot Filling," Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2021.
7. T. Pham, C. Tran, and D. Q. Nguyen, "MISCA: A Joint Model for Multiple Intent Detection and Slot Filling with Intent–Slot Co-Attention," Findings of the Association for Computational Linguistics: EMNLP, 2023.
8. B. Xing and I. W. Tsang, "Co-Guiding Net: Achieving Mutual Guidances between Multiple Intent Detection and Slot Filling via Heterogeneous Semantics-Label Graphs," Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2022.
9. Y. Chen and Z. Luo, "Pre-Trained Joint Model for Intent Classification and Slot Filling with Semantic Feature Fusion," Sensors, vol. 23, no. 5, 2023.
10. [10] M. Hardalov, I. Koychev, and P. Nakov, "Enriched Pre-Trained Transformers for Joint Slot Filling and Intent Detection," Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP), 2023.
11. [11] R. Gangadharaiah and B. Narayanaswamy, "Joint Multiple Intent Detection and Slot Labeling for Goal-Oriented Dialogue," Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT), 2019.
12. C.Nagarajan and M.Madheswaran - 'Stability Analysis of Series Parallel Resonant Converter with Fuzzy Logic Controller Using State Space Techniques'- Taylor &Francis, Electric Power Components and Systems, Vol.39 (8), pp.780-793, May 2011. DOI: 10.1080/15325008.2010.541746
13. C.Nagarajan and M.Madheswaran - 'Experimental verification and stability state space analysis of CLL-T Series Parallel Resonant Converter' - Journal of Electrical Engineering, Vol.63 (6), pp.365-372, Dec.2012. DOI: 10.2478/v10187-012-0054-2
14. C.Nagarajan and M.Madheswaran - 'Performance Analysis of LCL-T Resonant Converter with Fuzzy/PID Using State Space Analysis'- Springer, Electrical Engineering, Vol.93 (3), pp.167-178, September 2011. DOI 10.1007/s00202-011-0203-9
15. S.Tamilselvi, R.Prakash, C.Nagarajan,"Solar System Integrated Smart Grid Utilizing Hybrid Coot-Genetic Algorithm Optimized ANN Controller" Iranian Journal Of Science And Technology-Transactions Of Electrical Engineering, DOI10.1007/s40998-025-00917-z,2025
16. S.Tamilselvi, R.Prakash, C.Nagarajan," Adaptive sliding mode control of multilevel grid-connected inverters using reinforcement learning for enhanced LVRT performance" Electric Power Systems Research 253 (2026) 112428, doi.org/10.1016/j.epsr.2025.112428

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802233

17. S.Thirunavukkarasu, C. Nagarajan, 2024, “Performance Investigation on OCF and SCF study in BLDC machine using FTANN Controller," Journal of Electrical Engineering And Technology, Volume 20, pages 2675–2688, (2025), doi.org/10.1007/s42835-024-02126-w
18. C. Nagarajan, M.Madheswaran and D.Ramasubramanian- ‘Development of DSP based Robust Control Method for General Resonant Converter Topologies using Transfer Function Model’- Acta Electrotechnica et Informatica Journal , Vol.13 (2), pp.18-31,April-June.2013, DOI: 10.2478/aeei-2013-0025.
19. C.Nagarajan and M.Madheswaran - ‘DSP Based Fuzzy Controller for Series Parallel Resonant converter’- Springer, Frontiers of Electrical and Electronic Engineering, Vol. 7(4), pp. 438-446, Dec.12. DOI 10.1007/s11460-012-0212-0.
20. C.Nagarajan and M.Madheswaran - ‘Experimental Study and steady state stability analysis of CLL-T Series Parallel Resonant Converter with Fuzzy controller using State Space Analysis’- Iranian Journal of Electrical & Electronic Engineering, Vol.8 (3), pp.259-267, September 2012.
21. C.Nagarajan and M.Madheswaran, “Analysis and Simulation of LCL Series Resonant Full Bridge Converter Using PWM Technique with Load Independent Operation” has been presented in ICTES’08, a IEEE / IET International Conference organized by M.G.R.University, Chennai.Vol.no.1, pp.190-195, Dec.2007
22. Suganthi Mullainathan, Ramesh Natarajan, “An SPSS and CNN modelling based quality assessment using ceramic materials and membrane filtration techniques”, Revista Materia (Rio J.) Vol. 30, 2025, DOI: https://doi.org/10.1590/1517-7076-RMAT-2024-0721
23. M Suganthi, N Ramesh, “Treatment of water using natural zeolite as membrane filter”, Journal of Environmental Protection and Ecology, Volume 23, Issue 2, pp: 520-530,2022
24. T. He, X. Xu, Y. Wu, H. Wang, and J. Chen, “Multitask Learning with Knowledge Base for Joint Intent Detection and Slot Filling,” Applied Sciences, vol. 11, 2021.
25. L. Qin, F. Wei, T. Xie, X. Xu, W. Che, and T. Liu, “GL-GIN: Fast and Accurate Non-Autoregressive Model for Joint Multiple Intent Detection and Slot Filling,” Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL), 2021.
26. S. Zhang, Z. You, X. Qi, P. Liu, G. Wu, K. Xia, and S. Huang, “LCAN: A Label-Aware Contrastive Attention Network for Multi-Intent Recognition and Slot Filling in Task-Oriented Dialogue Systems,” Findings of the Association for Computational Linguistics: EMNLP, 2025.
27. Y. Song, J. Zhao, I. G. Harris, S. B. Koehler, and A. Abdullah, “PCMID: Multi-Intent Detection through Supervised Prototypical Contrastive Learning,” Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2022.
28. I. Vulić, I. Casanueva, G. Spithourakis, A. Mondal, T. Wen, and P. Budzianowski, “Multi-Label Intent Detection via Contrastive Task Specialization of Sentence Encoders,” Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2022.
29. Mathew, A., & Alex, H. (2025). Federated Learning for Secure Genomic Research: Privacy-Preserving AI Solutions for Precision Medicine. Science and Technology: Developments and Applications Vol. 9, 36-43.
30. Jagadeesh, S., & Soundappan, R. S. (2014). Survey on knowledge discovery in speech emotion detection. International Journal of Innovative Research in Computer and Communication Engineering, 2(5), 4476–4481. Retrieved from https://ijircce.com/admin/main/storage/app/pdf/i7mLTWLAd6a4VqXoYxeMRM6m0zylGcBFKaMTHo5H.pdf
31. Soundappan, S. J. (2022). AI-Based Fault Detection and Isolation for Reliability in Modern Power Systems. International Journal of Research Publications in Engineering, Technology and Management (IJRPETM), 5(4), 7106-7110.