LaneMate: A Hybrid Conversational Agent for Swim Session Management using LinearSVC and State-Based Dialogue Control

SANTHUSHA MALLAWATANTRI*, University of Nottingham, UK

Efficient management of swimming sessions requires a balance between rigid scheduling and flexible user interaction. This report presents *LaneMate*, a text-based conversational agent designed to automate session bookings and answer domain-specific queries. The system implements a **Hybrid Architecture**, combining a **LinearSVC** model utilising N-gram vectorisation for robust intent classification with a **Finite State Machine (FSM)** for secure, logic-driven transaction management.

Development followed an iterative approach, where the transition from Logistic Regression to LinearSVC improved intent classification accuracy from a baseline of 53% to 71% on the test set. Key features include state persistence for session continuity, context-aware interruptions allowing FAQ retrieval during transactional flows, and robust error recovery. The report concludes with an evaluation of system performance and usability, alongside a critical reflection on Responsible Research and Innovation (RRI), specifically focusing on data sovereignty and fairness in resource allocation.

CCS Concepts: • **Human-centered computing** → **Natural language interfaces**; *User studies*; • **Computing methodologies** → **Natural language processing**; *Discourse, dialogue and pragmatics.*

Additional Key Words and Phrases: Conversational Agent, Hybrid Architecture, LinearSVC, Finite State Machine, Usability Evaluation, RRI

1 Introduction

The management of recreational sports clubs is often subject to significant overheads, particularly regarding manual scheduling and member queries. While conventional Graphical User Interfaces (GUIs) offer structure, they lack adaptability to natural language, and human administrators are often resource-bound. This report presents **LaneMate**, a text-based conversational agent designed to fill this gap for a local swimming club.

LaneMate is a specialised, goal-oriented agent capable of handling two distinct interactions: transactional dialogue (multi-turn booking flows) and informational retrieval (answering Frequently Asked Questions about equipment and rules).

The primary technical challenge in developing LaneMate was balancing the flexibility of Natural Language Understanding (NLU) with the data integrity required for booking transactions. A pure machine learning solution risks hallucinating booking slots, whereas a pure rule-based solution can frustrate users with strict syntax. LaneMate addresses this with a **Hybrid Architecture**: it uses a **LinearSVC** machine learning model to probabilistically classify user intent, while delegating execution to a deterministic **Finite State Machine (FSM)** for the booking logic.

This report documents the development lifecycle of LaneMate. Section 2 describes the functional capabilities, technical implementation, and justification for the architectural decisions. Section 3 examines the conversational design principles implemented to ensure usability. Section 4 presents a quantitative and qualitative analysis of the system, and Section 5 reflects on ethical deployment, specifically regarding data privacy and bias within the scope of Responsible Research and Innovation (RRI).

Portfolio contact: hello@santhu.design hello@santhu.design

2 System Architecture

This section describes (a) the functional capabilities of LaneMate, (b) the technical implementation of its modules, and (c) the justification for the architectural decisions made during development.

2.1 Functional Capabilities

LaneMate implements the five core capabilities required of interactive NLP systems:

(1) **Intent Matching:** Classifies user input into 10 distinct intents (e.g., `book_session`, `cancel`).
(2) **Identity Management:** Acquires user names via the `set_name` intent and persists them explicitly alongside transaction records in `bookings.json`.
(3) **Transactional Booking:** Manages multi-turn dialogue to secure a slot (Type → Day → Time).
(4) **Information Retrieval:** Answers FAQs (e.g., "What kit do I need?") using vector similarity.
(5) **Small Talk:** Handles phatic communication (`greet`, `thanks`, `goodbye`) to maintain social rapport.

2.2 Architectural Overview

The system follows a modular design pattern. Figure 1 illustrates the high-level data flow. The system is implemented in Python using `scikit-learn` for the ML pipeline and standard JSON libraries for persistence.

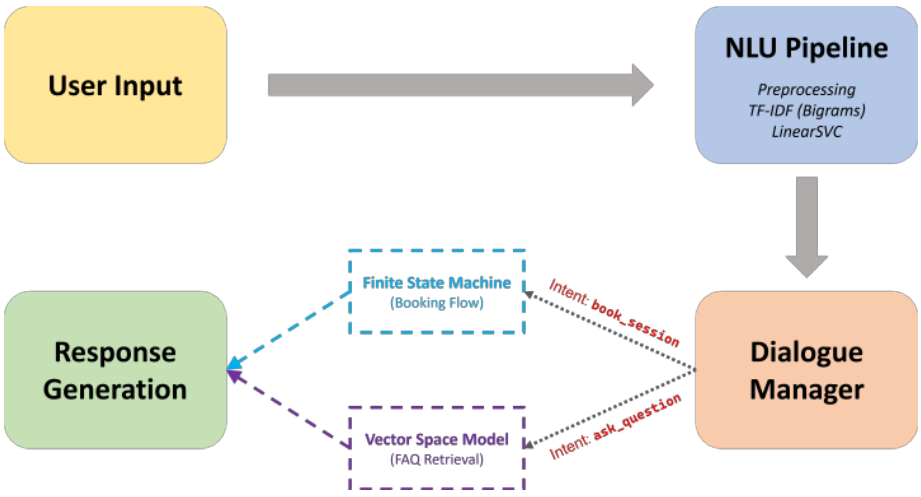

Fig. 1. LaneMate Hybrid Architecture: Integrating LinearSVC for NLU with a Finite State Machine for Dialogue Management.

2.3 Natural Language Understanding (NLU)

The NLU module acts as the entry point, converting raw user text into structured intents. The pipeline is constructed using `scikit-learn`'s `Pipeline` class to ensure consistent transformation of training and inference data.

2.3.1 Preprocessing.

Input text is normalised using the Natural Language Toolkit (`nltk`). Initially, the system employed standard stopword removal. However, testing revealed that for short command-based queries, stopwords often carry semantic weight (e.g.,

"*Who* is the coach" vs. "*Is* the coach here"). Consequently, the final preprocessing pipeline performs tokenisation and WordNet lemmatisation but deliberately retains stopwords to preserve directional meaning.

2.3.2 Feature Extraction and Vectorisation.

Text is converted into numerical vectors using Term Frequency-Inverse Document Frequency (TF-IDF). A critical design decision was the implementation of **Bigrams** (N-gram range 1,2). Early iterations using only Unigrams failed to distinguish between intents sharing core vocabulary (e.g., "*Book* a swim" vs. "*Cancel* booking"). By using Bigrams, the system captures "cancel booking" as a distinct feature, resolving the semantic overlap.

2.4 Evolution of the Intent Classifier

A core component of the development lifecycle was the iterative refinement of the classification algorithm. The project began with a Logistic Regression model, chosen for its baseline interpretability. However, as the intent definitions became more granular, the model struggled with the high-dimensional sparsity of the TF-IDF vectors.

Table 1 details the performance shift. To address the variance, the architecture was upgraded to a **Linear Support Vector Classifier (LinearSVC)**. LinearSVC minimises the squared hinge loss and is mathematically well-suited for sparse text data where decision boundaries are tight.

Table 1. Iterative Improvement of Intent Classification

Iteration	Architecture	Test Accuracy
Baseline	Logistic Regression (Unigrams)	38% - 53%
Refinement	LinearSVC (Bigrams, Balanced Weights)	**71%**

This architectural shift necessitated a critical implementation detail. Standard `LinearSVC` implementations in `scikit-learn` do not natively output probability distributions, which are required for the system's confidence threshold logic (Section 2.5). To resolve this, the classifier was wrapped in a `CalibratedClassifierCV` using Sigmoid calibration. This enables the pipeline to output a confidence score ($0.0 - 1.0$) for each prediction. This probability calibration was essential for distinguishing between "low-confidence booking attempts" (which should trigger a fallback) and "high-confidence small talk," stabilising the system's ability to distinguish between semantically similar intents.

2.5 Dialogue Management: The Hybrid Router

While NLU handles understanding, dialogue management handles action. LaneMate employs a custom router (implemented in the `handle_turn` method) that arbitrates control flow based on the current state. To prevent over-confident misclassifications, the router enforces a **confidence threshold of 0.25**; predictions below this value trigger a fallback help message.

2.5.1 Finite State Machine (FSM).

For booking transactions, the system enters a strict Finite State Machine (FSM) to ensure data integrity. The state transitions are defined as follows:

- `IDLE` $\rightarrow$ `GET_TYPE`: Triggered by the `book_session` intent. The system checks if a session type (e.g., "Squad") was extracted from the initial utterance. If yes, it skips directly to `GET_DAY`.

- `GET_TYPE` → `GET_DAY`: Validates that the user input matches a known session type (Squad/Lane/Lesson) before proceeding.
- `GET_DAY` → `GET_TIME`: Uses regex to extract a temporal entity. If the entity is valid, it is stored in the ‘slots‘ dictionary, and the state advances.
- `GET_TIME` → `CONFIRM`: Validates the time format. If valid, the system generates a summary string and requests explicit assent.
- `CONFIRM` → `IDLE`: On "Yes", the data is committed to ‘bookings.json‘. On "No", the transaction is aborted, and the slots are cleared.

This deterministic flow prevents the "hallucination" errors common in generative models, guaranteeing that a booking cannot be saved unless all required slots have passed validation.

Edge Case Handling: If a user attempts to change a slot mid-flow (e.g., "Actually, make it Tuesday"), the system overwrites the current slot variable. If the user provides an unrelated utterance (e.g., "Hello"), the FSM maintains the active state constraint and re-prompts for the missing slot, rather than resetting to IDLE, to prevent accidental data loss.

2.5.2 Smart Interruptions (Context Switching).

A limitation of pure FSMs is rigidity. To mitigate this, LaneMate implements a "Smart Interruption" logic. If the system is in a slot-filling state (e.g., `GET_DAY`) but the NLU detects an Information Retrieval intent (e.g., `ask_question`), the router pauses the FSM, serves the FAQ answer, and then re-prompts for the missing slot. This creates a fluid, non-blocking user experience.

2.6 Information Retrieval and Persistence

For general inquiries, the system utilises a Vector Space Model. User input is vectorised and compared against a knowledge base of FAQs using **Cosine Similarity**. A strict similarity threshold of 0.2 is enforced to prevent irrelevant answers.

The persistence layer is implemented as a flat list of booking objects in `bookings.json`. Each object stores the user name together with normalised day, time, and type. At query time, the program filters this list for matching user names. Although this is linear in the number of bookings ($O(n)$), at the expected scale of a small club this remains fast while keeping the implementation simple. The `check_duplicate_booking` helper reuses this filtered view to confine duplicate checks to the current user’s records, avoiding any need for a more complex database engine.

Finally, state persistence is managed via a local JSON document store (‘bookings.json‘). This ensures that user data survives session termination, fulfilling the requirement for a long-term memory component.

3 Conversational Design

Effective conversational agents must balance functional capability with usability. LaneMate was designed following core principles of conversational interface design, adhering to *Grice’s Maxims* [1] of Quantity and Relation to ensure navigability and user trust.

3.1 Prompt Design and Disambiguation

To minimise cognitive load, the system employs a *Slot-Filling* strategy with constrained prompts. Rather than using open-ended questions (e.g., “What do you want?”), the FSM utilises specific, directional prompts during transactions (e.g., “Which day?”). This **Disambiguation** strategy guides the user down the “Happy Path,” significantly reducing the

likelihood of Out-of-Vocabulary (OOV) errors. Additionally, lexical variation is implemented in the response generation (via `random.choice`), preventing the monotonous repetition often found in rule-based systems.

3.2 Discoverability and Affordances

A common barrier in text-based interfaces is the *Gulf of Execution*—users often do not know what actions are available [3]. LaneMate addresses this via a context-aware help system. If a user types "Help," the output changes dynamically based on the current FSM state (e.g., offering specific time examples only when in `GET_TIME`).

3.3 Error Handling and Repair

The system employs a two-tier error recovery strategy to manage *Graceful Degradation*:

(1) **Syntactic Validation:** The system uses Regular Expressions to sanitise critical slots (e.g., ensuring time inputs contain digits). If validation fails, the bot offers a corrective prompt ("That doesn't look like a time.").
(2) **Pragmatic Repair:** If the system fails to match an intent during a booking flow, the **Hybrid Fallback** logic checks if the user is asking a question, answering it, and then reprompting for the slot, preserving the conversational context.

3.4 Personalisation and Persistence

To foster long-term engagement, LaneMate maintains a persistent memory of the user's identity and schedule via a local JSON store. Upon initialisation, the system checks `bookings.json` and greets the user by name ("Welcome back Santhusha!").

3.5 Context Management

Beyond simple state tracking, LaneMate maintains a hierarchical context model. **Short-term context** is handled by the FSM, which remembers partially filled slots. **Long-term context** is handled by the JSON store, maintaining user history.

In practice, the "Smart Interruption" logic is implemented in a lightweight way: during slot filling, the FSM keeps its current state unchanged while temporarily handling FAQ turns, then resumes prompting for the same slot. This produces a stack-like interaction pattern (pause → answer → resume) without adding extra complexity to the state representation.

3.6 Confirmation and Control

Adhering to the principle of *Grounding*, the system implements an **Explicit Confirmation** step ("Please confirm: Squad on Monday at 6am? Yes/No") before committing data. Furthermore, in line with Nielsen's heuristic of *User Control and Freedom*, a global `cancel` command is available at any stage.

These design decisions are best illustrated by a typical booking scenario. A user might begin with a vague request such as "I want to swim sometime tomorrow." Instead of an open-ended query, the agent immediately narrows the problem space with a constrained slot prompt ("Sure, is that Squad or Lane?"). If the user later provides an ill-formed time (e.g., "18pm"), the system's regex validation triggers a targeted repair prompt ("That doesn't look like a time – please use something like 6am"), rather than a generic error. Crucially, the FSM retains the previously filled 'day' slot during this repair. In pilot testing, this combination of slot-filling prompts, local syntactic repair, and persistent state

significantly reduced conversational breakdowns, operationalising Grice's Maxim of Quantity by providing the user with just enough relevant information at each step.

4 Evaluation

The system evaluation employed a mixed-methods approach, triangulating quantitative system benchmarking with qualitative user usability testing.

4.1 System Performance

The training corpus was constructed with 260 labeled utterances distributed across 10 intents. The distribution was designed to be **non-uniform** to reflect real-world usage probabilities: transactional intents like `book_session` and `ask_question` were over-sampled (40 examples each) to capture higher lexical variance, while phatic intents like `greet` were kept smaller (15 examples). This class imbalance strategy forced the LinearSVC (via `class_weight='balanced'`) to penalise errors on complex intents more heavily during training, prioritising the core functionality over small talk.

The NLU classifier was developed using 5-fold stratified cross-validation during the tuning phase. For the final report, the model was evaluated on a held-out test set (20% of the corpus, N=52). Figure 2 presents the Confusion Matrix for the final **LinearSVC** model. The system achieved a weighted F1-Score of **0.72** and an overall accuracy of **71%**.

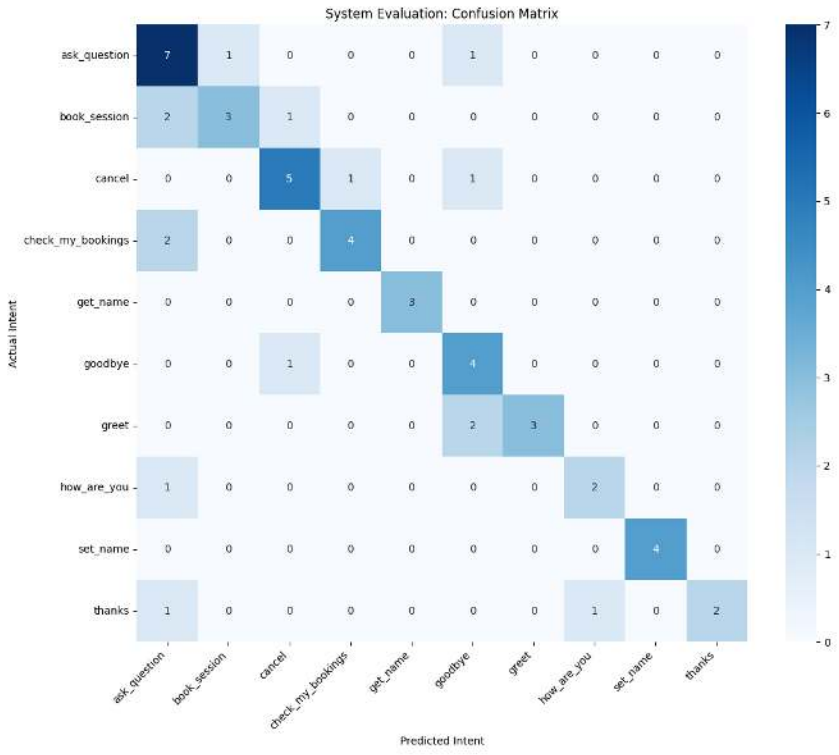

Fig. 2. Confusion Matrix showing strong diagonal density. Note the minor confusion between 'book_session' and 'ask_question'.

Table 3 provides a granular breakdown. Notably, the `set_name` and `get_name` intents achieved perfect precision (1.00), validating the model's ability to handle explicit identity commands.

4.1.1 *Misclassification Analysis.*

The lowest performance was observed in the `ask_question` intent (Precision 0.54). Table 2 highlights representative errors where short, ambiguous utterances confuse the boundary between inquiring and acting.

Table 2. Representative Classification Errors

User Utterance	Predicted	Actual
"What time is squad?"	book_session	ask_question
"Monday please"	ask_question	book_session

Critically, the Hybrid Architecture mitigates the impact of these errors. Once a user enters the booking FSM, the system effectively ignores NLU predictions for booking intents, "locking on" to the transaction. This ensures that even with 71% NLU accuracy, the transactional reliability remains high once initiated.

These metrics confirm the validity of the chosen architecture. The shift from a unigram Logistic Regression baseline to a bigram LinearSVC pipeline was particularly effective for intents that hinge on short lexical patterns, such as distinguishing "cancel booking" from "book a swim." Capturing two-word phrases allowed the classifier to exploit word order rather than treating tokens as an unordered bag. Furthermore, the alignment between 5-fold cross-validation scores during training and the held-out test accuracy suggests the model is not overfitting. The residual confusion around `ask_question` reflects genuine ambiguity in natural language phrasing rather than a purely technical failure. This finding directly motivated the implementation of the 0.25 confidence threshold and the deterministic FSM "lock-on" mechanism, ensuring that occasional intent classification noise does not cascade into incorrect financial transactions.

Table 3. System Performance Metrics (LinearSVC)

	precision	recall	f1-score	support
ask_question	0.54	0.78	0.64	9
book_session	0.75	0.50	0.60	6
cancel	0.71	0.71	0.71	7
check_my_bookings	0.80	0.67	0.73	6
get_name	1.00	1.00	1.00	3
goodbye	0.50	0.80	0.62	5
greet	1.00	0.60	0.75	5
how_are_you	0.67	0.67	0.67	3
set_name	1.00	1.00	1.00	4
thanks	1.00	0.50	0.67	4
accuracy			0.71	52
macro avg	0.80	0.72	0.74	52
weighted avg	0.76	0.71	0.72	52

4.2 User Usability Study

4.2.1 *Procedure.*

We recruited 3 participants (2 regular swimmers, 1 non-swimmer) via convenience sampling. Each participant interacted

with the system for approximately 10 minutes in a terminal environment. They were assigned high-level, goal-oriented tasks (e.g., "Book a session for Monday," "Check your bookings") rather than rote scripts, to test discoverability and error recovery in a natural setting. Metrics were derived from the *Chatbot Usability Questionnaire (CUQ)* [2], scored on a Likert scale (1-5).

Table 4. User Satisfaction Scores (1-5 Scale)

Metric	User 1	User 2	User 3	Avg
Navigability	5	4	5	**4.7**
Error Recovery	5	5	4	**4.7**
Naturalness	4	3	4	**3.7**

4.2.2 *Discussion of User Feedback.*

The high scores for **Navigability (4.7)** confirm the effectiveness of the constrained prompt design. Users specifically noted that the dynamic help messages prevented frustration when syntax errors occurred. The lower score for **Naturalness (3.7)** highlights the trade-off of the Hybrid approach: prioritising the **safety** of the booking transaction came at the cost of the **fluidity** typically associated with generative models.

Limitations: It is acknowledged that the small sample size ($N = 3$) and convenience sampling limit the statistical generaliability of these findings. Furthermore, the terminal-only interface may not reflect the experience of users accustomed to GUI-based apps. However, the qualitative insights align with the architectural trade-offs made during design.

5 Discussion and RRI

The development of LaneMate was guided by the **AREA Framework** [4] (Anticipate, Reflect, Engage, Act) to ensure Responsible Research and Innovation.

5.1 Application Area Reflection

In the context of a swimming club, the system acts as a gatekeeper to finite resources (lanes). **Fairness** is therefore critical. Given the 0.54 precision for `ask_question`, there is a risk that users with non-standard phrasing (e.g., non-native speakers) will be misunderstood, potentially leading to missed booking opportunities. The misclassification patterns observed in Section 4.1 fed directly into the *Reflect* and *Act* phases of the AREA framework, motivating the implementation of the confidence threshold and human-fallback strategies. Furthermore, storing attendance data for minors (Junior Squad) introduces risks of misuse (e.g., punitive monitoring of attendance), requiring strict governance and parental consent.

5.2 Privacy and Data Sovereignty

We **Anticipated** that storing user movement data constitutes a privacy risk. To **Act** on this, LaneMate implements a **Local First** architecture. User data is serialied into 'bookings.json' residing strictly on the client machine. This ensures **Data Sovereignty**—the user physically owns their data.

Reflecting on Evaluation: Given the 71% accuracy, misclassifications are inevitable. However, the explicit confirmation step and duplicate-booking checks (as detailed in Section 2) reduce the risk of unfair or unsafe allocations.

5.3 Bias and Inclusivity

Reflecting on the system's limitations, we acknowledge a significant **Linguistic Bias**. The training data is exclusively English. In a diverse university environment, this risks excluding international students. To **Engage** with this issue, future deployment would require informed consent procedures and the implementation of a clear "Human Fallback" option for complex queries.

The empirical results in Section 4 have direct implications for RRI. The lower precision of the `ask_question` intent (0.54), combined with user feedback regarding "robotic" interactions, suggests the model is tuned to "textbook" English rather than the fragmented phrasing often used by non-native speakers. In a club environment where lane access is a finite resource, this creates a fairness risk: members with non-standard dialects may be more likely to trigger fallback loops, leading to higher abandonment rates and under-representation in lane allocations. Within the AREA framework, these findings informed the 'Reflect' phase by quantifying linguistic bias. This motivated the 'Act' decision to design a clear human-fallback path for future deployment, ensuring that users with different language profiles are not systematically excluded from club resources.

6 Conclusion

LaneMate successfully demonstrates that a Hybrid Architecture—combining the flexibility of Machine Learning with the rigidity of Finite State Machines—is an effective solution for domain-specific transactional agents. By upgrading the classification pipeline to LinearSVC and implementing robust state management, the system achieved a 71% accuracy rate while ensuring the logical integrity of completed bookings. Future work should address the linguistic bias by integrating multilingual training sets. Architecturally, the next iteration should replace the static FAQ retrieval with a **Retrieval-Augmented Generation (RAG)** pipeline. By connecting the existing FSM guardrails to a Large Language Model (LLM), the system could generate more fluid, context-aware responses for open-ended queries while maintaining the deterministic safety required for database transactions.

References

[1] Herbert P Grice. 1975. Logic and conversation. *Speech acts* 41 (1975), 58–28.

[2] Simon Holmes, Anne Moorhead, Raymond Bond, Huiru Zheng, Vivien Coates, and Michael McTear. 2019. Usability testing of a healthcare chatbot: Can we use a standardized questionnaire? *Health Informatics Journal* 25, 4 (2019), 1211–1223.

[3] Jakob Nielsen. 1994. *Usability engineering*. Morgan Kaufmann.

[4] Jack Stilgoe, Richard Owen, and Phil Macnaghten. 2013. Developing a framework for responsible innovation. *Research Policy* 42, 9 (2013), 1568–1580.