# Chapter 2: Literature Review

## Contents

# 1 Introduction

In the context of informal service bookings in Pakistan, small businesses heavily rely on WhatsApp rather than formal software. Our system **BookForMe** is a bilingual (English / Roman Urdu), task-oriented booking agent built on a single application. This chapter reviews relevant literature in task-oriented dialogue, agentic systems, cross-lingual NLU, and retrieval-based planning, forming the foundation for our system design.

# 2 What constitutes an Agent?

While the term "agent" has historical roots in reinforcement learning and robotics, its application to Large Language Models (LLMs) represents a distinct shift in its association. Unlike static models that simply predict the next token, an agent is defined by its capacity for autonomous decision-making, persistent memory, and active interaction with its environment.

## 2.1 Defining the Agentic Architecture

Recent surveys have formally distinguished "Agents" from standard LLMs by their architectural components. Wang et al. [1] propose a unified framework where an Agent is not merely a language model, but a system composed of four distinct modules: **Profiling**, **Memory**, **Planning**, and **Action**.

In this framework, the LLM functions as the "Brain" or central controller. However, unlike a standalone model, an agent requires:

- **Perception:** The ability to perceive its environment (e.g., receiving a WhatsApp message or checking a database state).
- **Memory:** A persistent storage mechanism (Short-term and Long-term) to maintain context across asynchronous sessions, distinguishing it from the fixed context window of a standard LLM.
- **Action:** The capacity to execute tool calls (APIs) to impact the environment, rather than just outputting text [1].

This definition shifts the focus from simple "Text-Generation" to "Problem-Solving," where the system actively loops through a *Perception* → *Reasoning* → *Action* cycle.

## 2.2 The Shift from Chatbots to Agents

It is essential to differentiate between conversational chatbots and autonomous agents.

Traditional chatbots and conversational information systems are fundamentally *reactive*. As noted by Xi et al. [2], these systems function as static knowledge bases that respond solely to user prompts without maintaining an internal agenda. They excel at "Conversation" but lack the agency to initiate or manage "Workflows."

Furthermore, standard information systems (like RAG-based search engines) retrieve data but do not modify it. A chatbot might answer "What time is the salon open?", but an Agent must answer "Book the salon for 5 PM." The latter involves state changes in a database. This distinction is crucial for our system design, as it moves beyond the Read-Only limitations of traditional dialogue systems into Read-Write operations.

## 3 Frameworks for AI Agents

To implement the agentic architectures described above, researchers and practitioners utilize orchestration frameworks that abstract the complexity of prompt management, memory persistence, and tool invocation.

### 3.1 Cognitive Architectures for Reasoning

A defining characteristic of an agent is its ability to reason about its actions before executing them. Yao et al. [3] introduced the *ReAct* paradigm, which fundamentally enables this by interleaving chain-of-thought reasoning with external actions. This allows the model to plan: *"Thought: User wants to book $\rightarrow$ Action: Check Database $\rightarrow$ Observation: Slot available"*.

However, simple ReAct loops can suffer from error propagation. Liu et al. [4] address this in the **PRACT** agent by incorporating a "self-reflection" step. Before executing a tool call (e.g., confirming a booking), the agent internally verifies if the action aligns with the user's intent. This reflective capability is critical for the agent to verify any action that would result in a permanent change in the database.

### 3.2 Orchestration Frameworks: From Chains to Cyclic Graphs

The implementation of autonomous agents requires a robust middleware to manage the interaction between the LLM "Brain" and external tools. **LangChain** has emerged as a widely adopted framework for this purpose. As highlighted by Mavroudis [10], LangChain provides a modular abstraction layer that decouples the application logic from specific LLM providers, allowing you to combine different, unrelated components (e.g., prompt templates, vector stores, and output parsers) into unified pipelines.

However, traditional implementations often rely on Directed Acyclic Graphs (DAGs), or "Chains," where data flows linearly from input to output. While sufficient for simple

Retrieval-Augmented Generation (RAG), linear chains struggle with the iterative nature of complex agents.

To address this, the **LangGraph** framework extends the LangChain ecosystem by shifting from DAGs to **Cyclic State Graphs**. In this paradigm, the agent's behavior is modeled as a state machine where nodes represent actions (e.g., "Check Availability") and edges represent conditional logic (e.g., "If slot full, go to Suggest-Alternative"). This graph-based approach aligns with the "Reflexion" architectures discussed by Shinn et al. [12], where an agent must persist its state across multiple iterations to refine its output.

LangGraph allows us to define the agent's logic as a state machine. Unlike a linear chain that runs from A to Z, a graph can loop back: Start → Check Payment → Invalid? → Ask Again → Check Payment. This ”cyclic” capability is essential for WhatsApp interactions, where users might reply late, send the wrong image, or change their mind halfway through. Shang et al. [9], in their paper discussing AgentSquare, highlight that modular architectures like this, separating Planning, Reasoning, and Memory—are significantly more robust for complex tasks than monolithic prompts.

For our use case, LangGraph provides the necessary infrastructure to handle the non-linear nature of WhatsApp booking conversations, where users may change their minds, provide partial information, or require confirmation loops.

## 4 Task-Oriented Dialogue and State Tracking

The core functionality of any booking agent relies on Task-Oriented Dialogue (TOD) systems, which have evolved from modular pipelines to end-to-end architectures. A foundational baseline in this domain is **SimpleTOD**, proposed by Hosseini-Asl et al. [15]. This model treats dialogue state tracking as a unified causal language modeling task, jointly predicting belief states, actions, and responses. While effective, such end-to-end approaches typically require extensive supervised datasets, creating a barrier for specialized, low-resource domains like informal service bookings in Pakistan.

To address the challenge of data scarcity, Shin et al. [16] introduced the **Dialogue Summaries as Dialogue States (DS2)** framework. Rather than maintaining complex belief states, DS2 converts conversation history into templated summaries that map directly to structured representations. This summarization-based approach is particularly useful, as it simplifies state tracking for short, noisy messages without requiring the massive annotated corpora demanded by traditional deep learning models.

The complexity of our specific domain is further compounded by the linguistic nature of the input, which involves frequent code-switching between English and Roman Urdu. Yu et al. [17] address multilingual state tracking through contrastive learning, which helps align slot embeddings across different. However, this method requires significant data to be a viable. Building on this, Wu et al. [18] demonstrate that parameter-efficient adapter methods can effectively transfer models to code-switched contexts. These techniques allow the system to handle mixed-language inputs without the computational cost of training a multilingual model from scratch.

Finally, processing Roman Urdu presents unique challenges due to a lack of standardized orthography, a single word like "waqt" (time) may be written as "vakt", "wkht", or "tym" [20]. Bhat et al. classify this as *orthographic variation*, noting that standard subword tokenizers often fail to map these variants to a single semantic token. By integrating the adapter-based strategies proposed by Wu et al. [18], our design could employ a specialized normalization layer before intent classification, ensuring that phonetic variations are resolved into consistent temporal slots despite spelling inconsistencies.

To address these multilingual and orthographic challenges efficiently, an alternative approach is to leverage state-of-the-art multilingual language models like Qwen2.5, which demonstrate exceptional performance on multilingual tasks without the need for complex custom adapters or large-scale annotated data. Specifically, the Qwen2.5-7B model achieves a score of 79.3 on the Multi-Understanding benchmark, the highest among models of similar size, showcasing its robustness in handling diverse linguistic contexts such as code-switching between English and Roman Urdu [19]. We adopt Qwen2.5 as the backbone for our TOD system to simplify state tracking and response generation. Its inherent multilingual capabilities help normalize orthographic variations and align semantic representations across languages. This approach reduces computational overhead and data requirements compared to traditional adapter-based methods.

## 4.1 Data Augmentation for Low-Resource Conversational Agents in Booking Systems

Building accurate natural language understanding (NLU) components for WhatsApp-based receptionists is challenging due to the limited availability of annotated dialogues in informal service domains. In our context, intents such as checking slot availability, selecting services, or submitting payment proofs often appear in code-switched English-Roman Urdu, and real-world labelled datasets rarely exceed a few hundred examples per intent. Collecting and annotating additional dialogues is both time-consuming and costly, creating a bottleneck for supervised learning.

Data augmentation has proven effective in addressing such low-resource scenarios for task-oriented dialogue systems. Hou et al. [22] demonstrated that back-translation and contextual augmentation can improve slot-filling F1 scores by 6–17% and intent accuracy by up to 12% in booking domains with fewer than 500 annotated utterances. Controllable T5-based generation techniques yield 8–18% absolute gains in intent classification for restaurant and hotel reservation tasks under 10-shot conditions [23], while fully synthetic dialogues generated via large language models can surpass real-data baselines, achieving 4–12% higher Joint Goal Accuracy on MultiWOZ when human annotations are extremely limited [24].

Guided by these insights, **BookForMe** leverages LLM-driven synthetic dialogue generation and paraphrasing to produce thousands of diverse training examples. These include salon/spa-specific service requests, slot inquiries, and payment confirmations in both English and Romanised Urdu. This augmentation strategy enhances intent recognition robustness, mitigates sparsity in low-resource settings, and ensures reliable performance for real-world WhatsApp interactions with informal service vendors.

# 5 Retrieval, Planning, and Tool Use in Autonomous Agents

## 5.1 Retrieval-Augmented Generation (RAG)

Lewis et al. [21] introduce **RAG**, which retrieves relevant documents and conditions the generator on them. RAG integrates a retrieval module with a generative language model to improve factual correctness and context-awareness.

In the context of BookForMe, RAG enables the agent to access vendor information, availability, pricing, and slot status dynamically from Firestore. This approach ensures that the system does not rely solely on memorized knowledge in the model parameters, which is crucial for up-to-date booking information in informal service settings.

## 5.2 Adaptive Retrieval and Neuro-Symbolic Integration

Modern conversational systems emphasize adaptive retrieval strategies [25], where the agent determines when and what to retrieve. Unlike static retrieval pipelines, adaptive retrieval uses the agent's internal reasoning to minimize unnecessary queries, reduce latency, and improve accuracy.

This capability aligns naturally with a neuro-symbolic architecture. In our project, the LLM functions as a reasoning core, while deterministic tools execute booking functions. The agent can plan, retrieve vendor availability, and confirm bookings with minimal intervention. Integrating symbolic reasoning ensures that database actions remain transactional and idempotent, preventing double bookings or inconsistent states.

## 5.3 Tool Use and API-Oriented Agent Architectures

Toolformer [5] demonstrates that LLMs can self-learn to decide when to call external APIs. Similarly, Gorilla [6] and ToolLLM [8] explore controlled invocation of structured functions for web and API-based tasks.

BookForMe leverages these ideas to orchestrate its core functions:

- **get_availability(vendor, date):** Fetches real-time availability from Firestore.
- **create_booking(user, vendor, slot):** Confirms a booking atomically with concurrency safeguards.
- **get_services(vendor):** Retrieves services and pricing dynamically to populate user-facing menus.

This integration ensures that the agent maintains autonomy while adhering to transactional safety, a feature not fully addressed by prior agentic frameworks.

### 5.4 Planning and Decision-Making in LLM Agents

Beyond retrieval and tool invocation, agents require explicit planning mechanisms. ReAct [3] and PRACT [4] highlight the importance of interleaving reasoning with action. In practice, this means the agent must:

1. Evaluate user input and extract intents/slots.
2. Determine which retrieval actions or tool calls are necessary.
3. Plan sequences of actions, accounting for constraints like vendor availability or user preferences.
4. Execute actions while performing internal validation to prevent errors.

Graph-based orchestration frameworks such as LangGraph allow modeling these sequences as cyclic state graphs, enabling the agent to retry, self-correct, and handle asynchronous conversations, which is critical in WhatsApp-based bookings.

## 6 Multi-Agent architecture

As Wu et al. [11] argue in their analysis of multi-agent frameworks, effective problem-solving often requires *loops*, the ability to retry a failed action, request missing information, or self-correct based on feedback. Recent work by Handler (2023) proposes a comprehensive multi-dimensional taxonomy for autonomous LLM-powered multi-agent architectures, highlighting how complex, real-world tasks can benefit significantly from decomposing workflows across specialized agents rather than relying on a single monolithic model [7]. In their framework, agents are assigned distinct roles (e.g., intent parsing, verification, database updates, external communications), and a central coordinator manages orchestration, collaboration, and error recovery. This design enables handling of asynchronous interactions, retries, and conditional logic which are all essential for robust conversational booking through WhatsApp, where users may provide incomplete information, incorrect payment proofs, or require follow-up. For BookForMe, adopting such a modular architecture allows us to isolate concerns (e.g., slot-holding, receipt verification, vendor approval), improving maintainability, scalability, and fault-tolerance. By referencing Handler's taxonomy as conceptual underpinning, we substantiate our choice of a multi-agent design using frameworks such as LangGraph, rather than a simpler single-agent pipeline.

## 7 Gap Analysis

To clearly contextualize the contributions of BookForMe, we compare it against existing paradigms in Table 1.

Table 1: Comparison of BookForMe against existing dialogue paradigms.

| **System Type** | **Tool Use** | **Roman Urdu** | **Async Memory** | **Focus** |
|---|---|---|---|---|
| Standard Chatbots (ChatGPT) | ✓ | ✓(Partial) | × | General Chat |
| Traditional TOD (SimpleTOD) | × | × | × | Research datasets |
| Agent Frameworks (ReAct) | ✓ | × | ✓(Limited) | Web Automation |
| **BookForMe (Ours)** | ✓ | **✓(Optimized)** | ✓ | **WhatsApp Booking** |

The primary gaps identified are:

- **Roman Urdu code-switching**: Existing DST research does not target informal WhatsApp-style Roman Urdu, presenting significant gaps in language modeling and normalization.
- **Transactional booking guarantees**: While agent tool-use is well studied, very few works address concurrency, idempotency, and transactional safety in real-world booking systems.
- **Asynchronous conversation flow**: Most TOD frameworks assume synchronous interaction. WhatsApp requires persistent memory and session resumption.
- **Evaluation for real-world SMEs**: Agent evaluation surveys do not address booking systems in informal economies where reliability and low latency are essential.

## 8 Conclusion

The reviewed literature supports a bilingual, tool-using, memory-enabled agent capable of autonomous reasoning and booking operations. However, specific gaps including code-switching, transactional safety, and robust asynchronous agent behavior directly motivate our system design. BookForMe synthesizes these strands into a practical LLM-driven booking agent optimized for real use in Pakistan's small business ecosystem.

## References

[1] Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., ... & Wen, J. R. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 1-26. DOI: 10.1007/s11704-024-40231-1.

[2] Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., ... & Gui, T. (2023). The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*. `https://arxiv.org/abs/2309.07864`.

[3] Yao, S. et al. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv preprint arXiv:2210.03629*. `https://arxiv.org/abs/2210.03629`.

[4] Liu, Z. et al. (2024). PRACT: Optimizing Principled Reasoning and Acting of LLM Agents. *arXiv preprint arXiv:2410.18528*. `https://arxiv.org/abs/2410.18528`.

[5] Schick, T. et al. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. *arXiv preprint arXiv:2302.04761*. `https://arxiv.org/abs/2302.04761`.

[6] Patil, V. et al. (2023). Gorilla: Large Language Model Connected with APIs. *arXiv preprint arXiv:2305.15334*. `https://arxiv.org/abs/2305.15334`.

[7] Händler, T. (2023). Balancing autonomy and alignment: A multi-dimensional taxonomy for autonomous LLM-powered multi-agent architectures. *arXiv preprint arXiv:2310.03659*. `https://arxiv.org/abs/2310.03659`.

[8] Qin, Y. et al. (2023). ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. *arXiv preprint arXiv:2307.16789*. `https://arxiv.org/abs/2307.16789`.

[9] Shang, Y. et al. (2024). AgentSquare: Automatic LLM Agent Search in Modular Design Space. *arXiv preprint arXiv:2410.06153*. `https://arxiv.org/abs/2410.06153`.

[10] Mavroudis, Vasilios. (2024). LangChain. *Preprint*. DOI: 10.20944/preprints202411.0566.v1.

[11] Wu, Q. et al. (2023). AutoGen: Enabling Next-Gen LLM Applications. *arXiv preprint arXiv:2308.08155*. `https://arxiv.org/abs/2308.08155`.

[12] Shinn, N. et al. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. *Advances in Neural Information Processing Systems*, 36. `https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf`.

[13] Yehudai, A. et al. (2025). Survey on Evaluation of LLM-based Agents. *arXiv preprint arXiv:2503.16416*. https://arxiv.org/abs/2503.16416.

[14] Ji, Z. et al. (2024). Testing Erroneous Planning in LLM Agents. *arXiv preprint arXiv:2404.17833*. https://arxiv.org/abs/2404.17833.

[15] Hosseini-Asl, E. et al. (2020). SimpleTOD: End-to-End Task-Oriented Dialogue. *arXiv preprint arXiv:2005.00796*. https://arxiv.org/abs/2005.00796.

[16] Shin, J. et al. (2022). Dialogue Summaries as Dialogue States (DS2). *Findings of the Association for Computational Linguistics: ACL 2022*. https://aclanthology.org/2022.findings-acl.302/.

[17] Yu, D. et al. (2023). Cross-lingual Dialogue State Tracking via Contrastive Learning. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. https://aclanthology.org/2023.ccl-1.54/.

[18] Wu, Y. et al. (2023). Towards Zero-Shot Multilingual Transfer for Code-Switched Responses: An Adapter-based Cross-lingual Framework. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL 2023 Long Papers)*. https://aclanthology.org/2023.acl-long.417/.

[19] Qwen Team (2025). Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*. https://arxiv.org/abs/2412.15115.

[20] Bhat, I. et al. (2023). A Survey of Code-Mixed NLP: Methods and Challenges. *arXiv preprint arXiv:2510.07037*. https://arxiv.org/abs/2510.07037.

[21] Lewis, P. et al. (2020). Retrieval-Augmented Generation (RAG). *arXiv preprint arXiv:2005.11401*. https://arxiv.org/abs/2005.11401.

[22] Hou, Y., Che, W., Lai, Y., Liu, Z., & Liu, T. (2020). Few-shot Slot Tagging with Collapsed Dependency Transfer and Label-enhanced Task-adaptive Projection Network. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 1381–1393. https://aclanthology.org/2020.acl-main.127/

[23] An, Z., Zhang, L., Wang, Y., & Zhao, D. (2022). Controllable Data Augmentation for Few-Shot Spoken Language Understanding. *Proceedings of INTERSPEECH 2022*, 4182–4186. https://www.isca-archive.org/interspeech_2022/an22_interspeech.html

[24] He, W., Dai, H., Li, S., Zhang, M., Jin, L., & Liu, T. (2024). SynthDST: Synthetic Data is All You Need for Few-Shot Dialog State Tracking. *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 11658–11673. https://aclanthology.org/2024.emnlp-main.684/

[25] Chen, Z. et al. (2025). Adaptive Retrieval-Augmented Generation for Conversational Systems. *Findings of the Association for Computational Linguistics: NAACL 2025.* https://aclanthology.org/2025.findings-naacl.30.pdf.