

سلام خولي

Salam snd Yusra report.docx

 Midocean University

Document Details

Submission ID

trn:oid::2945:409493058

Submission Date

Aug 26, 2026, 7:05 PM GMT+3

Download Date

Aug 26, 2026, 9:37 PM GMT+3

File Name

Salam snd Yusra report.docx

File Size

7.0 MB

67 Pages

15,463 Words

92,402 Characters

22% detected as AI

The percentage indicates the combined amount of likely AI-generated text as well as likely AI-generated text that was also likely revised by AI.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.



33 AI-generated 22%

Likely generated or likely generated and revised by AI.

Disclaimer

Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely human generated as AI generated and likely AI generated as human generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.



MIDOCEAN UNIVERSITY
COLLEGE OF INFORMATICS
MASTER OF SCIENCE IN ARTIFICIAL INTELLIGENCE

**AI Chatbot with an Integrated Product Recommendation
System for E-Commerce Platforms**

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Artificial Intelligence

Prepared by

Researcher	Student ID
Yusra Elfaran	2021202562
Salam Khouli	261000006

Supervised by

Dr. Khaled Eskaf

College of Informatics, Midocean University
Moroni, Comoros

2026

Approval Page

This thesis, entitled “Design and Implementation of an Intelligent AI Chatbot with an Integrated Product Recommendation System for E-Commerce Platforms,” was prepared and submitted by Yusra Elfaran and Salam Khouli in partial fulfillment of the requirements for the Master of Science in Artificial Intelligence at the College of Informatics, Midocean University.

The thesis focuses on developing an intelligent AI chatbot that works together with a product recommendation system to provide users with a more convenient and personalized e-commerce experience. The project demonstrates the practical use of artificial intelligence in improving the way users interact with online shopping platforms.

After review, this thesis is approved as meeting the academic requirements for the Master of Science in Artificial Intelligence degree.

Supervisor: Dr. Khaled Eskaf

Examiner 1: _____

Examiner 2: _____

Date: _____

Declaration of Originality

We declare that this thesis presents the work undertaken for the stated master's project and that all external ideas, methods, and publications used in the manuscript are acknowledged through in-text citation and the reference list. The experimental results, screenshots, software artifacts, and model files reported in the thesis are drawn from the supplied project archive and the executed verification activities. This thesis has not been submitted in the same form for another degree.

Yusra Elfaran: _____

Salam Khouli: _____

Acknowledgements

The researchers thank Dr. Khaled Eskaf for his guidance helpful comments and focus on clear methods, during the whole project. Appreciation is also extended to the College of Informatics at Midocean University for providing the academic framework in which the study was completed. The researchers are grateful to their families and colleagues for their encouragement during the implementation, experimentation, and documentation stages.

الملخص

الأهداف: تصميم وتنفيذ وتقييم روبوت محادثة ذكي للتجارة الإلكترونية يدمج تصنيف النوايا بالبحث عن المنتجات والانحدار اللوجستي، وتوثيق TF-IDF بخط أساس يعتمد على DistilBERT والتوصية القائمة على المحتوى، مع مقارنة كتجربة خفيفة منفصلة. المنهجية: اتبعت الدراسة تصميمًا تجريبيًا تطبيقيًا. نُظمت بيانات النوايا من BERT-Tiny عبارة إلى 11,586 عبارة موزعة على 20 فئة، وقُسمت طبقًا إلى 8,110 للتدريب، و1,738 للتحقق، 14,500، NVIDIA Tesla T4، باستخدام وحدة معالجة رسومات Google Colab على DistilBERT و1,738 للاختبار. دُرّب وربط بكتالوج SQLite وFastAPI وReact وقورن بخط الأساس على مجموعة الاختبار نفسها. نُفذ التطبيق باستخدام يضم 8,000 منتج. وشمل التحقق النهائي 12 سيناريو محادثة مختارًا، ومجموعة انحدار وظيفية من 52 حالة، و15 فحصًا نهائيًا للتشغيل والواجهات البرمجية والأصول المحلية وسلامة البيانات والأخطاء المنطقية. النتائج: حقق BERT-مرجحة 0.9564، مقابل 0.8993 و0.8978 لخط الأساس. وحقق F1 دقة 0.9563 ودرجة DistilBERT دقة 0.8548 في تجربة منفصلة لا تمثل مقارنة معمارية مضبوطة. نجحت حالات الانحدار الـ 52 والفحوص Tiny التشغيلية الـ 15 دون إخفاق بعد تصحيح عيوب التنفيذ، بما فيها الاعتماد الخارجي لصورة الهيرو، تهيئة قاعدة البيانات، التحقق من المدخلات، واستقلال تشغيل المودلين محليًا. اقتصر تقييم التوصية على مؤشرات تشخيصية بديلة دون أحكام بشرية أو سجلات تفاعل حقيقية؛ لذا لا تثبت النتائج التخصيص أو رضا المستخدم. التوصيات: توثيق مصدر البيانات، وتكرار التدريب ببذور متعددة، وجمع محادثات حقيقية بموافقة المستخدمين، وتحسين بيانات المنتجات والأسعار، وإجراء اختبارات الأمن والحمل والاستعادة قبل النشر.

الكلمات الدالة: روبوت محادثة للتجارة الإلكترونية، تصنيف النوايا، DistilBERT، التوصية القائمة على المحتوى، التجارة الحوارية.

Abstract

Objectives: I aim to design, implement and evaluate an e-commerce chatbot that combines classification, product search and recommendation. I will compare DistilBERT with a TF-IDF logistic-regression baseline. I will record a separate lightweight test of BERT-Tiny.

Methodology: An applied experimental design was used. Cleaning reduced 14,500 intent utterances to 11,586 examples across 20 classes, stratified into 8,110 training, 1,738 validation, and 1,738 test examples. DistilBERT was trained on Google Colab with an NVIDIA Tesla T4 GPU and compared with the baseline on the same test set. The application used React, FastAPI, SQLite, and an 8,000-product catalog. Final verification comprised 12 selected conversational scenarios, a 52-case functional regression suite, and 15 focused runtime, API-validation, recommendation, and hidden-issue checks.

Results: DistilBERT achieved 0.9563 accuracy and 0.9564 weighted F1, versus 0.8993 accuracy and 0.8978 weighted F1 for the baseline. BERT-Tiny achieved 0.8548 accuracy in a separate experiment that was not treated as a controlled architectural comparison. All 52 regression cases and all 15 focused runtime, API, recommendation, and hidden-issue checks passed after correcting implementation defects, including the external hero dependency, database initialization order, input validation, and offline model loading. Recommendation evaluation used proxy diagnostics without human relevance judgments or real interaction logs; therefore, the findings do not establish personalization or user satisfaction.

Recommendations: I suggest documenting where the data comes from retraining with several random seeds gathering real conversations that have user consent improving the completeness of product and price data and running security, load and recovery tests before launching the system.

Keywords: e-commerce chatbot, intent classification, DistilBERT, content-based recommendation, conversational commerce.

Table of Contents

Approval Page	i
Declaration of Originality	ii
Acknowledgements	iii
Abstract	v
Table of Contents	vi
List of Figures	ix
List of Tables.....	x
List of Abbreviations.....	xi
1.1 Background of the Study	1
1.2 Research Problem	2
1.3 Aim and Objectives	3
1.4 Research Questions.....	4
1.5 Significance of the Study.....	4
1.6 Scope, Boundaries, and Limitations	4
1.7 Research Contributions.....	7
1.8 Thesis Organization	7
1.9 Chapter Summary	7
2.1 Introduction	9
2.2 Theoretical Foundations	9
2.2.1 Transformer Self-Attention.....	10
2.2.2 TF-IDF and Logistic Regression	11
2.2.3 Content-Based Recommendation.....	11
2.3 E-Commerce Chatbots and Intent Classification.....	12
2.4 Conversational Recommendation.....	13
2.5 Hybrid Statistical and Rule-Based Systems	13
2.6 Previous Empirical Studies.....	13
2.7 Comparative Analysis and Research Gap	14
2.8 Chapter Summary	16
3.1 Research Design	17
3.2 System Requirements	17
3.3 Data Resources and Preprocessing	18
3.3.1 Primary Intent Dataset	18
3.3.2 Auxiliary BERT-Tiny Dataset Pipeline	19
3.3.3 Product Catalog.....	19
3.4 Model Training	19
3.4.1 Controlled Baseline.....	19

3.4.2 DistilBERT Fine-Tuning	20
3.4.3 BERT-Tiny Auxiliary Experiment.....	20
3.5 Product Retrieval and Recommendation	20
3.6 Proposed System Architecture.....	21
3.6.1 UML Sequence Model.....	22
3.6.2 Use-Case Model.....	23
3.6.3 Data-Flow Model	24
3.7 Software Implementation	24
3.7.1 Frontend	24
3.7.2 Backend and API	25
3.7.3 Database.....	25
3.7.4 Chat Routing and Business Logic.....	26
3.8 Evaluation Protocol	26
3.9 Reproducibility, Ethics, and Privacy	26
3.10 Chapter Summary	27
4.1 Dataset and Training Results	28
4.2 Controlled Classification Results	29
4.3 Per-Class Analysis	30
4.4 Auxiliary BERT-Tiny Results	32
4.5 Recommendation Diagnostics	33
4.6 Chatbot and API Experiments	34
4.6.1 Executed Chatbot Scenarios.....	34
4.6.2 Automated API Verification	36
4.6.3 Interface Verification	37
4.6.4 Final Runtime Verification and Hidden-Issue Corrections	37
4.7 Discussion of Research Questions.....	42
4.8 Limitations and Threats to Validity.....	44
4.9 Chapter Summary	45
5.1 Conclusion	46
5.2 Summary of Contributions	47
5.3 Recommendations	47
5.3.1 Data	47
5.3.2 Modeling	47
5.3.3 Recommendation	48
5.3.4 Deployment.....	48
5.3.5 Multilingual Extension.....	48
5.4 Future Work Roadmap	48

5.5 Final Statement	49
References	50
Appendix A: Intent Labels and Operational Meanings.....	52
Appendix B: Executed Chatbot Scenario Matrix	53
Appendix C: Final Runtime Validation	53
Appendix D: Installation and Execution Guide	55
D.1 Backend	55
D.2 Frontend.....	55
D.3 Verification	55

List of Figures

Figure 1 Proposed System Architecture	21
Figure 2 sequence diagram for store	22
Figure 3 use-case diagram for store.....	23
Figure 4 1 data-flow diagram level one.....	24
Figure 5 Distribution of the 20 intent classes after primary data cleaning.....	28
Figure 6 DistilBERT training and validation behavior across four epochs.....	29
Figure 7 Controlled comparison between TF-IDF logistic regression and DistilBERT.	30
Figure 8 DistilBERT F1-scores	31
Figure 9 DistilBERT confusion matrix on test.....	31
Figure 10 Rating and price distributions.	33
Figure 11 six representative queries proxy metrics.....	33
Figure 12 Executed chatbot tests for product, typo, policy, and out-of-scope requests.....	35
Figure 13 home, catalog-search, product interfaces.....	37
Figure 14 research summary screen	37
Figure 15 homepage screen.....	40
Figure 16 shopping chat	40
Figure 17 product-comparison	41
Figure 18 negative-budget validation message	41
Figure 19 out-of-catalog response	42

List of Tables

Table 1 Research questions	4
Table 2 Comparison studies	14
Table 3 Functional requirements	17
Table 4 Non-functional requirements.....	18
Table 5 Data resources	18
Table 6 data cleaning and split	19
Table 7 Model config	20
Table 8 components and responsibilities for store	21
Table 9 API endpoints.....	25
Table 10 Evaluation measures.....	26
Table 11 classification performance for test	30
Table 12 model results	32
Table 13 BERT-Tiny results	32
Table 14 Recommendation diagnostics.....	34
Table 15 Test part A.....	34
Table 16 Test part B	35
Table 17 Test application verification.....	36
Table 18 Final implementation.....	38
Table 19 question findings	42
Table 20 Future-work points	48
Table 21 -B.1 Executed chatbot scenarios	53
Table 22 - C.1 Final runtime, API, asset, and hidden-issue validation results.....	53

List of Abbreviations

Abbreviations used in the thesis

Abbreviation	Meaning
AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CPU	Central Processing Unit
DFD	Data Flow Diagram
F1	Harmonic Mean of Precision and Recall
GPU	Graphics Processing Unit
JSON	JavaScript Object Notation
NLP	Natural Language Processing
NLU	Natural Language Understanding
REST	Representational State Transfer
RQ	Research Question
SQL	Structured Query Language
SQLite	Embedded Relational Database Engine
TF-IDF	Term Frequency-Inverse Document Frequency
UI	User Interface
UML	Unified Modeling Language

CHAPTER 1

INTRODUCTION

1.1 Background of the Study

There are e-commerce websites and they often use old ways to search, compare and suggest products based on what users like or do. They use set rules and algorithms. Some stores have chatbots that use rules and answers. These are not better at understanding what users want. Our project will be completely different. Users will write what they want to buy in the chat. The system will figure out what they want and show the right products based on that. For example a user might want a product compare two items or see the best products—all during a chat on the website.

Understanding what the user wants is the part of this. Transformer models (Vaswani et al., 2017) gave us a way to look at sentences as a group of words of just a mix of words. BERT improved this by training in both directions. Showed how changing the model can make classification much better (Devlin et al. 2019). DistilBERT made it smaller and cheaper. Still kept most of the language understanding (Sanh et al. 2019).. Lets be honest: a high score on a test does not mean the chatbot is helpful. The classifier must work with the way the system sends users to the place finds products uses a database and shows the results.

Recommending products during a conversation adds another difficulty. Recent studies (Gao & Jannach 2021) show that understanding the conversation finding what the user likes how to recommend and how to check everything are all connected. Our project focuses on a version: we classify what the user wants decide what to do and sort products from a local list using TF-IDF with some changes to the data. The app is made with a React/Vite interface, a FastAPI server, an SQLite database and ready-to-use BERT models with logic that keeps responses inside the systems limits.

Earlier studies have shown that any virtual assistant needs a way to know things an easy-to-use interface and a way to connect requests with services (Mohamed et al. 2020). And that is what we did: we used learning classification with fixed rules and app services. Why? Because just being accurate in theory is not enough. Mistakes can come from finding the words, from the database or even how the answer is shown. On the hand sometimes the

system fixes the mistake from the rules and gives the right answer. So we check the model on its own and the app as a whole.

The study contributes an reproducible prototype and evaluates it at three levels. First the study reports a controlled comparison between TF-IDF regression and DistilBERT on the same cleaned intent dataset and the same held-out test split. Second the study reports BERT Tiny because that model was trained using a pipeline that cleaned and sampled data differently. Third the study checks how the live application behaves by using API tests. It runs chat scenarios. This separation matters because evaluating the model checking recommendations and testing software function answer questions and should not be mixed into one accuracy statement.

The system we built is for e-commerce of beauty products. The list shows names, descriptions, ratings, some prices, images and other details to help users choose. The chatbot can do twenty types of tasks such as searching for products getting information, comparing items setting a price limit adding items to a cart placing an order paying, arranging delivery finding store information giving feedback and even handling orders that are not covered.

1.2 Research Problem

Studies on e-commerce chatbots often focus on one part of the system. Some studies look at how the system can detect what the user wants and report how accurate the model is. They do not explain how the model works in a store. Others show an interface. Do not show how well it understands what users say. This creates a gap between how the model performs and how the app works in life. A model might do well on a test. Act differently when connected to real business tasks. Misspelled words, ways to talk about prices, follow-up questions and messages outside the topic can make the app choose a path even if the model scores high. The opposite can happen too. The app might seem to use a machine learning model. The main decisions are made with rules or fixed words. In this case just showing the model's score does not explain why the app made a choice.

It is also important to explain how the model that identifies what the user wants the rules that decide what to do the way products are found and the services, in the app all work together. It is also important to make sure that each part is clear when checking how the system works.

Another issue comes from the two models we used. They were not trained in the way so their results can't be directly compared. The DistillBERT model cleaned the data removed repeated samples and used 11,586 statements in a 70/15/15 split.

The BERT-Tiny model took a path using 12,851 clean samples. So the two models have the number of records and examples but one is faster and better for the CPU, than the other.

Because of these issues the main research problem is figuring out how to bring a BERT-based model that finds what the user wants the way products are found the rules that direct the path and the store services together while making sure that the machine learning results and how the app works are clear and can be checked. So the study should check not what the model predicts but also how the app uses these predictions and how the other parts of the system affect the final answer.

1.3 Aim and Objectives

The main goal of this study is to design, build and test an e-commerce chatbot that brings together intent classification, product search based on content and database-backed application features. The specific objectives are :

1. To record the e-commerce intent dataset how it was cleaned, the spread of classes and the split, into stratified training, validation and test sets.
2. To fine-tune DistilBERT for multiclass e-commerce intent classification and compare its performance with a TF-IDF logistic-regression baseline using the same processed dataset and held-out test set.
3. To evaluate overall and class-level classification performance using accuracy, weighted and macro precision, recall, F1-score, learning curves, and the confusion matrix.
4. To report the BERT-Tiny model as a separate lightweight auxiliary experiment without presenting its result as a controlled architectural comparison with DistilBERT.
5. To implement an integrated application that maps predicted intents to product search, content-based recommendation, product information, price filtering, policy, cart, order, and customer-support functions.
6. To evaluate the running chatbot through executed user scenarios and automated API tests while distinguishing raw intent predictions from final application routes.
7. To identify the data, recommendation, experimental, privacy, and deployment limitations that affect the interpretation and generalization of the study results.

These objectives are addressed in the Results and Discussion section through dataset and training analysis, controlled model comparison, class-level error analysis, the separate BERT-

Tiny report, recommendation diagnostics, executed chatbot scenarios, API verification, and critical analysis of the study limitations.

1.4 Research Questions

RQ1. How accurately does DistilBERT classify the 20 predefined e-commerce intents compared with a TF-IDF logistic-regression baseline evaluated on the same processed dataset and held-out test split?

RQ2. Which intent classes remain the most difficult to classify, and what error patterns are revealed by the class-level metrics and confusion matrix?

RQ3. How does the deployed chatbot translate a raw model prediction into a final application route for product search, recommendation, product information, policy, cart, order, and customer-support functions?

RQ4. What do the executed chatbot scenarios and API tests reveal about raw intent predictions, deterministic routing refinement, response behavior, functional completion, and recorded latency?

RQ5. What data, recommendation, experimental, and deployment limitations prevent the current results from being interpreted as evidence of personalized recommendation quality or production-level effectiveness?

Table 1 study question

Research Question	Primary Evidence	Thesis Location
RQ1	Controlled DistilBERT vs. TF-IDF experiment	Table (11), Figures (6)-(7)
RQ2	Per-class report and confusion analysis	Table (12), Figures (8)-(9)
RQ3	Architecture, routing, and implementation trace	Figures (1)-(4), Tables (8)-(9)
RQ4	Executed chat scenarios and automated API tests	Tables (15)-(18), Figures (12)-(20)
RQ5	Recommendation diagnostics and methodological limitations	Table (14), Figures (10)-(11), Section 4.8

1.5 Significance of the Study

The importance of the study partly lies in the way the study separates and examines the data that was collected. The project uses results from than one method but the study does not treat these results as if they came from the same setup. The study compares models separately

from the help experiment while the study presents classification results on their own not linked to the software test. Likewise the study treats recommendation measurements as indicators of how retrieval works not as proof of what users like. The study therefore ensures that the study does not say more than what the experiments show.

The evaluation is not about how accurate the classifier's overall. The evaluation also looks at each category. The evaluation is helpful because the categories of what people want are not all the same and some are very similar. The evaluation gives an idea of what the classifier's good at and where the classifier might have problems. The same idea applies to parts of the system. The offline measurements show how well the system works on stored data while the recommendation checks show how items are found and ordered. The functional tests make sure that the messages reach the place in the software. These different types of evidence answer questions and are presented separately.

User studies add another kind of evidence by looking at things such as how helpful how trustworthy and how satisfied people're. The study was not part of the project. The study does not consider the recommendations and outcomes as proof of user satisfaction. Explaining this limitation helps others repeat and check the results using the notebooks, model files, test results and other items in the project archive.

The study also has real-world importance because the study creates a chatbot that works as a working example, not a model that was trained. Natural language questions are connected to product records, store rules, shopping cart actions, orders and product pages that users see. The system can run without needing a paid language model service making the system good for examples and small tests in conversation-based shopping.

However the experiments show that the system does not rely completely on the intent classifier. In some cases the system needed to fix spelling mistakes and use rules before it worked best. So the real value is not just in using a trained model. The study mixes ideas with clear business rules. The study lets us see how each part affects the result. The study helps to know if the good response comes from the model from rules or from both together.

1.6 Scope, Boundaries, and Limitations

The main focus is on task-based conversation in shopping: finding products suggesting items, comparing items moving items to the cart handling orders, delivering, paying, helping with accounts giving feedback and knowing when something is not related. The study does not look at conversations. Making up new responses.

The language used is for understanding English. Arabic appears in the abstract and the keywords. The classifier was not checked on mixed-language requests.

The data used includes files, trained models, an SQLite index with 8,000 products, notebooks, measurements, screenshots and test reports found in the project archive. The original source and how the data was labeled are not in the archive. This is a problem, not a conclusion.

The experiments look at a fixed comparison between TF-IDF and DistillBERT regression on the same group. The BERT-Tiny report was made using a way of cleaning and checking so it is more of a description than a real comparison.

The software used is a test version running locally. Some parts were. The tests showed that the functions worked during the study.

The tests do not show how the application would work as a service. Security for production was not tested. Were scalability, how easy it is for people to use or how well it works over time. Privacy needs for a system with customer data would also need a separate check before it can be used.

The test used data from the project and messages made by the researchers not real customer talks. It is not said that the system is ready to take or keep information. For any use we must only take what is needed obtain permission and keep data under strict control. Deciding who can see the data set limits on how data's kept is a must. We must make sure that messages are not identifiable. Checking trust is part of the process. Keeping logs of interactions carts and orders is essential. Using user talks to train the model without permission and without following rules is forbidden. These rules matter because when messages have no names, the paths and trust scores can reveal how people behave.

User talks must not be used to train the model without permission and, without following the rules. The range of products is important. Out of 8,000 product records only 2,534 have a price. This means 5,466 records or 68.3 % do not have any price. Therefore price filtering tests look at part of the products. Cannot be seen as a full look at the whole catalog. Also the catalog is mostly in the "All Beauty Products" group, with a few high-end beauty items so the results cannot be used for all online stores.

1.7 Research Contributions

- A comparison between TF-IDF and DistillBERT logistic regression using the same filtered data and the same test group.
- A clear difference between the DistillBERT way and the separate BERT-Tiny path.
- A complete e-commerce chatbot that uses intent prediction with search, suggestions, rules, cart orders and support.
- A way to check the idea and the final way the system works including the reasons for going over the original plan.
- A three-step way to check the model the recommendations and the real software parts.
- An explanation of the problems with the data, the index not having checks and not testing how well it works.

1.8 Thesis Organization

This thesis has five chapters. Chapter 1 introduces the study. Explains the research problem, goals, questions why it matters, what it covers and what it adds. Then the research behind the project is discussed. Chapter 2 looks at the work on Transformers. Intent classification along with conversation suggestions, not related topics and systems that use both. These topics give the background needed for the way this study is made. Chapter 3 explains what was done in the project. It covers the data. How it was prepared how the models were made how products were found, the system design and how the software was made. The way of checking was also explained in this chapter. Chapter 4 shows the results. How well the model worked was looked at along with the recommendations the chatbot tests and the API check. This chapter connects the evidence to the research questions of looking at each result on its own. Chapter 5 ends the study by talking about what the results show, what the study added and what can be done next. Extra information that would stop the discussion is in the appendices. This includes the list of what people want detailed test cases, a summary of the API check and how to run the system.

1.9 Chapter Summary

This chapter talked about the problem that led to making the e-commerce chatbot. Explained what the study is trying to look into. The goals and questions for the study were set, along with the limits that the results should be seen in. A key point in this chapter is that the chatbot is not just tested by how accurate its model's. The study also looks at what happens

after an intent is guessed, including the rules and services that decide the path. This difference is important for the experiments that come later. The chapter also found limits in the data. How some tests could be compared. These limits are kept when looking at the results not as problems that're separate, from the test. Now Chapter 2 looks at the research and the ideas that helped make the system.

CHAPTER 2

RELATED WORK

2.1 Introduction

This chapter looks at ideas and earlier work that're most relevant to the system created in this project. The review focuses on four areas: contextual language representation intent classification in e-commerce, dialogic recommendations and hybrid routing that mixes inferred predictions, explicit rules and structured knowledge. The discussion covers the ideas that shaped the system we built well as the limits that stay in earlier studies.

2.2 Theoretical Foundations

Transformer architecture replaced processing with self-attention. Each token can weight tokens in the sequence. This allows training (Vaswani et al., 2017).BERT applies bidirectional Transformer encoding. BERT is pre-trained through masked-language modeling and sentence-level objectives before fine-tuning (Devlin et al. 2019).

In classification a classification head is trained over the contextual representation of an utterance.The benefit over bag-of-words methods is that word meaning can depend on surrounding terms and ordering. However performance still depends on label quality, domain coverage and similarity among definitions.

DistilBERT is a BERT-family model produced through knowledge distillation. DistilBERT original study reported a 40% size reduction and faster inference while retaining most of BERT's language-understanding performance (Sanh et al. 2019).

These properties make DistilBERT a candidate for a research prototype that runs inference on a local CPU after GPU-based training.BERT-Tiny represents an aggressive reduction in model capacity.

In the project BERT-Tiny serves as an optional low-resource model. However BERT-Tiny experimental result cannot isolate architecture effects because preprocessing, sample size and optimization settings differ from the pipeline.A traditional TF-IDF model represents documents using term importance based on within-document frequency and corpus-level rarity (Salton & Buckley 1988).

Combined with regression TF-IDF provides an interpretable and computationally inexpensive baseline.

Such a baseline is important because transformer should be justified by gains on the same test data rather than by model novelty alone. In this study the controlled baseline uses unigram and bigram TF-IDF features, class-balanced logistic regression and the exact primary test split used for DistilBERT. Intent classification is operationally defined as assigning one of 20 predefined labels to a user utterance. The final route is the application action selected after deterministic refinement.

These are not always identical: raw intent is the model prediction whereas the final route reflects the hybrid system decision. Functional case completion means that the application returned the expected type of behavior for a selected test prompt. It is not an estimate of model accuracy over a sampled population. Aly, Eskaf and Selim (2017) also demonstrate the value of expert-system design in decision-support applications. Rule-based. Structured knowledge translate user inputs into actionable recommendations. This supports the study's use of transparent routing safeguards alongside statistical intent prediction.

The recommendation component is content based because the project archive does not contain user item interaction histories. Content-based methods rank items by looking at how similar query terms are to product descriptions. These content-based methods can also use item metadata such as rating and popularity (Adomavicius & Tuzhilin 2005). The Amazon Reviews 2023 resource gives us review and item metadata at a scale. This is the source for the product catalog used by the project (Hou et al. 2024).

In the system I built TF-IDF similarity finds what is relevant. I use rating, review count and a stored recommendation score to help with reranking. This design is easy to reproduce. I cannot learn preferences because I do not have user behavior data. For this study chatbot implementation means the path from a user message to a result you can see on the screen. That path includes normalization, classifier inference, deterministic refinement, business function routing, database or recommendation access response construction and frontend rendering. I check reliability by using held-out model evaluation and deterministic functional tests. To make a system reliable for production you would also need user studies, repeated performance trials, load testing, security assessment, monitoring and evaluation on data collected after deployment. I left those tasks outside the study boundaries.

Detecting things that're out of scope is very important for a commerce assistant. A closed intent inventory cannot safely handle every request. Larson et al. (2019) Showed that

classifiers that work on in-scope categories can still struggle with inputs. In this system, out of scope is both a learned label and an application boundary. It stops the chatbot from acting like it knows things outside of shopping order, delivery, payment and support functions. People who study dialogue system evaluation also warn us not to rely on a score.. (Deriu et al 2021)

Show the difference between component level measurements end to end testing and human evaluation. This difference is why I use a three layer protocol: intent classification, recommendation diagnostics and executed software scenarios. I report each layer separately because each layer measures a thing.

2.2.1 Transformer Self-Attention

The main Transformer operation works by calculating attention weights between token representations. To keep it simple you get the attention output by turning the input into query, value matrices and then using scaled dot-product attention. This mechanism lets each token grab information, from parts of the sentence. This is very helpful when the intent of a sentence depends on how product terms, actions and constraints all connect to each other.

$$Attention(Q, K, V) = softmax(QK^T / \sqrt{d_k})V \quad (1)$$

2.2.2 TF-IDF and Logistic Regression

TF-IDF is still a starting point because TF-IDF turns documents into simple vectors that are easy to train. If you look at a term *t* in document *d* the weight of that term goes up when the term appears a lot in one place. However the weight of that term goes down if that term shows up in documents. After that Logistic regression looks at the feature vector to guess the chance of each intent label. I like this baseline because it is easy to understand. It helps us see if using a Transformer actually makes a difference.

$$TF-IDF(t, d) = TF(t, d) \times \log(N / DF(t)) \quad (2)$$

2.2.3 Content-Based Recommendation

Content-based retrieval puts the user request and the product text, into the vector space. The engine I used looks at cosine similarity to see how well the words match. Then the engine mixes that similarity score with title matching, full-text matching and a stored recommendation score. This way of working does not need to know a user history to work.. This method cannot figure out what a person likes over a time. Content-based recommendation also has a time if the user request and the product use words.

$$\text{cosine}(q, p) = (q \cdot p) / (\|q\| \|p\|) \quad (3)$$

$$\begin{aligned} \text{combined score} &= 0.50 \text{ similarity} + 0.30 \\ &\text{title score} + 0.10 \text{ text score} + 0.10 \text{ stored} \\ &\text{recommendation score} \end{aligned} \quad (4)$$

2.3 E-Commerce Chatbots and Intent Classification

Task-oriented chatbots in e-commerce are different from open-scale systems because the functions they support are clearly listed. The assistant has to decide if a message is asking for a search, a recommendation a comparison, a price selection, an action in the shopping cart an order service, an explanation of policy or a support reply. Intent classification helps in evaluating the system. Also brings boundary problems. Categories that are semantically similar can. Requests that are not supported may be forced into the nearest known category unless handling for out-of-scope requests is deliberately built.

Sharma (2020) showed that context between phrases can improve classification in an e-commerce voice assistant. This discovery is important because later messages for example "Show me products " cannot be understood correctly without the earlier product request. Typical stores today are limited to a structured context that includes recent products and chosen items but their classifiers are trained on single phrases. For these reasons contextual classification stays an addition.

Kulkarni et al. (2023) Focused on classification of conversational intents using classifiers and pointed out how hard it is to handle grammatical errors and requests in multiple languages. The current implementation uses standardization of spelling mistakes and checks selected inputs that contain spelling errors. However the training data set is still in English and the project does not use unclassified production conversations. Consequently this study gives evidence of practicality. Does not solve the lack of classifiers or the robustness of a multilingual system.

Larson et al. (2019) Showed that detecting out-of-range messages is a challenge from classifying in-range messages. Even classifiers that work well still confidently put messages into intents. The current system uses out of range classification and fixed limits, for claims that're clearly irrelevant. However a broader real world evaluation needs unsupported requests and carefully set rejection thresholds.

2.4 Conversational Recommendation

Examined recommendation from angles for example how user preferences are inferred how conversations are understood and how recommendation strategies are chosen and evaluated. Janash et al. (2021) Also pointed out that conversational recommendation depends on components rather, than only a ranking method. In the system this recommendation idea is used in a way. Users can describe the product they want. Add conditions or constraints in natural language. The backend then finds products and rearranges the results based on the available product information.

Hu et al. (2024) Presented 2023 Amazon reviews and language representations of items for retrieval and recommendation. The project index is part of this set of resources. Shows how product titles, descriptions, ratings and review counts can help local retrieval. However the used index is a aesthetically focused subset and the exact conversion process from the public source to the SQLite database is not fully described in the submitted archive. This source-related limitation is stated of being guessed from assumptions.

Evaluation is essential. Coverage and average similarity can show that the engine returns candidates, but they do not prove that users think these candidates are appropriate, diverse, useful or satisfactory. Therefore human judgments about how suitable the candidates are about interaction outcomes are needed to make stronger claims. This study therefore shows the recommendation results as diagnoses.

2.5 Hybrid Statistical and Rule-Based Systems

The combination of prediction and explicit decision logic follows expert systems research. The system also mixes prediction with explicit decision rules. This is the idea used in expert systems, where results that are learned or inferred can be supported by knowledge and predefined logic. The chatbot combines prediction with rules. The model first identifies the user intent while application rules can adjust the route when needed. Ali et al. (2017) Showed how structured knowledge and fuzzy inference can help with decisions and Mohammed et al. (2020) Noted that a domain specific chatbot also needs a knowledge structure and a routing mechanism. These ideas support the design used in this project. These principles are the foundation of the architecture, where the model provides an intent and deterministic safeguards check expressions such as pricing, payment terms, shipping terms and product keywords before selecting the final path.

Hybrid design has two advantages. First it can protect business functions from errors that're easy to spot through clear patterns. Second it provides an audit log that shows why the final path is different from the prediction. However this design also has a risk: if only the final path is reported the rules may hide errors made by the model. Therefore this thesis records both values and treats the overshoot rate as an indicator rather, than a precise metric.

2.6 Previous Empirical Studies

Table 2 Comparison studies

Study	Focus	Method/Data	Key Finding	Gap Relative to This Study
Devlin et al. (2019)	BERT pre-training and fine-tuning	Large pretrained Transformer evaluated across NLP tasks	Bidirectional contextual representation improves downstream performance	Not domain-specific and does not address application integration
Sanh et al. (2019)	DistilBERT compression	Knowledge distillation from BERT	Reduced size and faster inference with retained capability	Does not evaluate e-commerce routing or recommendation
Larson et al. (2019)	Intent and out-of-scope evaluation	Large intent dataset with unsupported requests	Out-of-scope detection remains difficult	No e-commerce application or business-function integration
Sharma (2020)	E-commerce voice-assistant intent classification	Context-enhanced classifier	Inter-utterance context improves follow-up interpretation	No product recommendation or full-stack verification
Mohamed et al. (2020)	Virtual assistant for academic advising	Knowledge structure, interface, and routing	Demonstrates applied assistant integration	Different domain and no Transformer comparison
Gao et al. (2021)	Conversational recommender systems	Survey of methods and challenges	Evaluation must include dialogue and recommendation dimensions	Survey; does not provide a directly comparable artifact
Jannach et al. (2021)	Conversational recommendation	Systematic survey	Recommendation, dialogue management, and interaction design are interdependent	No single implementation benchmark
Kulkarni et al. (2023)	Label-efficient intent classification	Semi-supervised conversational classification	Addresses limited labels and noisy user language	No integrated store or recommendation engine
Hou et al. (2024)	Language-item retrieval and Amazon Reviews 2023	Large product/review resource and retrieval models	Shows rich language-item representations	Current project uses a smaller classical local subset

2.7 Comparative Analysis and Research Gap

Sharma in 2020 looked at classification in an English language e commerce voice assistant. Sharma discovered that context between phrases helps in understanding follow up requests. This study is very important because shopping talks usually have follow up requests. For example a filter request that comes after a product search. The current model keeps a context like recent product IDs and items chosen but its classifier uses just one phrase. This discrepancy suggests a need for expansion in versions: contextual intent models should evaluate intents rather than relying primarily on the state of the command.

Kulkarni et al. (2023) Addressed classification of conversational intents using labels. Kulkarni study highlighted challenges posed by errors and multilingual user queries. Kulkarni findings support the need to test more than grammatically correct phrases. The current project includes spell correction and implementation of prompts. However the training and testing dataset is only in English. Unlike semi-supervised approaches this method does not exploit interaction records. Therefore the current project provides evidence while leaving the classification efficiency and robustness of the multilingual system unresolved.

Gao et al. (2021) Structured the challenges of recommendation systems around preference induction, multi-role dialogue strategy, dialogue comprehension, exploration and evaluation. Janash et al. (2021) Demonstrated that dialogic recommendation combines recommendation, dialogue management and interaction design. These studies highlight limitations of evaluating recommendations through similarity. The implemented prototype can respond dialogically. Display labeled products. However the prototype does not learn long-term preferences. Refine the dialogue strategy based on interaction outcomes. Therefore the prototype metrics should be described as tools for retrieval.

Hu et al. (2024) Presented Amazon 2023 reviews and language-item representations for retrieval and recommendation. Hu study illustrates the scope and richness that can be achieved by linking metadata and review language. The current project uses a subset of product metadata focusing on aesthetics. The project also uses a TF-IDF classifier instead of a trained semantic retrieval. This simplification aids implementation. However the absence of prices and lexical ambiguity in the catalog limits the quality of price filtering and product type matching.

The research gap that the current project addresses lies not in a converter architecture but in the integration and accurate evaluation of a small-scale e-commerce chatbot model. Current studies often focus on improving intent models or conversational recommendation methods. The current study connects an evaluated classifier to a practical application. The study demonstrates the contribution of guidance and clearly distinguishes between controlled classification evidence, assisted model outcomes recommended alternatives and software validation. This framework provides a description of what the implemented system has and has not proven.

Larson et al. (2019) Presented a dataset on intents explicitly including out-of-scope queries and demonstrated that rejecting unsupported requests is more difficult than classifying supported intents. Larson finding is relevant to the out-of-scope" category. Larson study also

notes that this category was the lowest-rated category according to the F1 scale in the DistillBERT test report. However the current study differs by linking the classifier to product order and policy actions in a prototype working store.

Derrieux et al. (2021) Conducted a survey to evaluate dialogue systems. Confirmed that automated metrics do not fully reflect user experience or task success. Janash et al. (2021). Gao et al. (2021) Also indicated that dialogue recommendations should be evaluated based on dialogue quality, recommendation usefulness and interaction outcomes. These findings justify the papers rejection of interpreting retrieval indicators or specific completion scenarios as indicators of user satisfaction.

The resulting research gap lies in an verifiable e-commerce prototype where model accuracy, rules-based routing, product retrieval and application validation are reported within clearly defined experimental boundaries. This contribution does not represent a Transformer architecture. Instead this contribution offers a framework for implementation and evaluation that demonstrates how a trained intent model functions within a complete application and highlights evidence gaps supporting claims related to production.

This study expands upon the approach by combining the chatbot routing principles from Mohammed et al. (2020) With the consistent decision support logic from Ali et al. (2017). Both principles are applied to an e-commerce chatbot and a content-based product retrieval context.

Therefore the research gap lies in an verifiable evaluation framework and not, in a new neural architecture. The thesis presents a controlled DistillBERT experiment.

2.8 Chapter Summary

we have set up the theory behind contextual intent classification, lightweight Transformer models, content-based retrieval and hybrid routing. The papers I looked at show how useful these methods are but they also show that we must keep classifier metrics, recommendation evidence and end-to-end behavior separate. Those findings guide the methodology and the proposed framework that I outline in Chapter 3.

CHAPTER 3

METHODOLOGY AND PROPOSED FRAMEWORK

3.1 Research Design

The study used an applied software-engineering design. The unit of analysis for the machine-learning experiment was an English e-commerce utterance paired with one of twenty intent labels. The unit of analysis for product retrieval was a catalog record. The unit of analysis for verification was an executed user message or automated API assertion. This design permits evaluation at model, retrieval and application levels while maintaining boundaries between them.

The project archive, trained model folders, notebooks, SQLite database, application source code, configuration files, metrics and screenshots were treated as the research artifacts. Results were accepted only when they could be traced to these artifacts or reproduced by executing the code. When provenance or experimental detail was missing the thesis pointed out the limitation of guessing.

3.2 System Requirements

Table 3 Functional requirements

ID	Requirement
FR-01	Accept a natural-language shopping or support message through the web interface.
FR-02	Normalize common spelling variations and extract explicit constraints such as price values.
FR-03	Predict one of twenty supported e-commerce intents and return a confidence value.
FR-04	Apply deterministic routing safeguards and record any override reason.
FR-05	Search the product catalog by textual content and product terms.
FR-06	Generate content-based product recommendations and rerank candidates.
FR-07	Filter products by a user-specified maximum price when price data are available.
FR-08	Return product details and compare selected products.
FR-09	Support simulated cart, checkout, order, payment, delivery, and support functions.
FR-10	Detect and respond to requests outside the supported store domain.
FR-11	Expose evaluation metrics and model information through backend endpoints.

ID	Requirement
FR-12	Render structured JSON responses as text, product cards, actions, and updated context.

Table 4 Non-functional requirements

ID	Quality Attribute	Requirement
NFR-01	Traceability	Record raw intent, final route, confidence, and override reason.
NFR-02	Local execution	Core intent and retrieval functions operate without a paid external generative-model API.
NFR-03	Maintainability	Separate frontend, API, model service, recommendation service, and database layers.
NFR-04	Reproducibility	Retain model configurations, metrics, test reports, and executable source code.
NFR-05	Performance	Return interactive responses within the local test environment; no production SLA is claimed.
NFR-06	Privacy	Use research data and researcher-defined messages; no private customer dataset is used.
NFR-07	Security boundary	Simulated payment and order actions must not be presented as production transactions.
NFR-08	Usability	Provide product cards, clear feedback, and accessible routes to search, compare, and act.

3.3 Data Resources and Preprocessing

Table 5 Data resources

Resource	Size	Role in the study	Qualification
Primary intent corpus	14,500 raw; 11,586 cleaned	Controlled baseline and DistilBERT experiment	Original source and annotation process are not documented in the archive
Auxiliary intent pipeline	12,851 cleaned; 3,939 capped training cases	BERT-Tiny deployment experiment	Different cleaning, sampling, and split; descriptive only
Product catalog	8,000 product records	Search, filtering, and content-based recommendation	5,466 records have no usable price
Functional chat set	12 prompts tested with two models	Raw prediction, final routing, response, and latency inspection	Purposefully selected cases; not a random accuracy sample
Automated application suite	15 assertions	Endpoint and business-scenario verification	Does not cover security, load, usability, or recovery

3.3.1 Primary Intent Dataset

The primary notebook identified four fields: id, text, label, and set, with twenty raw labels. The cleaning procedure normalized whitespace, removed empty records, retained utterances between 3 and 300 characters, removed rows matching a documented profanity/noise pattern,

removed duplicate text-label pairs, and retained labels with at least twenty samples. This process reduced the primary corpus from 14,500 to 11,586 utterances, removing 2,914 rows. The cleaned dataset was encoded and split using stratification with a fixed random seed of 42 into 8,110 training, 1,738 validation, and 1,738 test samples.

Table 6 data cleaning and split

Stage	Rows	Description
Raw intent data	14,500	20 intent labels
Cleaned data	11,586	79.9% retained; 2,914 records removed
Training set	8,110	70.0% of cleaned data
Validation set	1,738	15.0% of cleaned data
Test set	1,738	15.0% of cleaned data

3.3.2 Auxiliary BERT-Tiny Dataset Pipeline

A second notebook produced a cleaned dataset of 12,851 utterances and a 70/15/15 split of 8,995 training 1,928 validation and 1,928 test records. Training was capped at 200 examples per intent resulting in 3,939 training examples used by BERT-Tiny. Because this pipeline differs from the dataset in cleaning, sampling and split its result is reported as auxiliary deployment evidence rather than a controlled comparison.

3.3.3 Product Catalog

The SQLite catalog contains 8,000 product records: 7,905 in All Beauty and 95 in Premium Beauty. Average rating is 4.19 the median rating is 4.3. The average review count is 486.46. 2,534 Products contain a usable positive price; 5,466 products have missing price values. The product catalog supports search, filtering and content-based recommendation. The missing-price rate limits the candidate pool for budget requests.

3.4 Model Training

3.4.1 Controlled Baseline

The baseline uses unigram and bigram TF-IDF features with a vocabulary of 50,000 minimum document frequency of two and class-balanced logistic regression. The model is trained on the 8,110 primary training examples and evaluated on the same 1,738 held-out test examples used for DistilBERT. This design makes the comparison controlled with respect to the data split.

3.4.2 DistilBERT Fine-Tuning

The primary model is distilbert-base-uncased. Utterances are tokenized to a maximum sequence length of 96. Training uses four epochs, a training batch size of 16 an evaluation batch size of 32 and a learning rate of 2×10^{-5} . The supplied notebook records training on Google Colab using an NVIDIA Tesla T4 GPU. Validation loss and classification metrics are monitored after each epoch and the final model folder contains configuration, tokenizer, label maps and safetensors weights.

3.4.3 BERT-Tiny Auxiliary Experiment

The auxiliary model uses prajjwall/bert-tiny with a maximum sequence length of 64, ten epochs, batch size 16 learning rate 5×10^{-4} and a cap of 200 training examples per intent. The archive says this model can be used and it does not take a lot of computer resources. Its test results are not used to compare the architecture with DistilBERT because the data system's different.

Table 7 Training Configuration

Approach	Main configuration	Training/Test	Interpretation
TF-IDF + logistic regression	TF-IDF 1–2 grams; max 50,000; min_df=2; class-balanced; seed 42	8,110 / 1,738	Controlled baseline
DistilBERT	distilbert-base-uncased; length 96; 4 epochs; batch 16/32; LR 2×10^{-5}	8,110 / 1,738	Controlled primary model
BERT-Tiny	prajjwall/bert-tiny; length 64; 10 epochs; batch 16; LR 5×10^{-4}	3,939 / 1,928	Separate auxiliary pipeline

3.5 Product Retrieval and Recommendation

At the start of the app the recommendation service loads the SQLite product table. It fits a TF IDF vectorizer to the product_text field that has, up to 8,000 features. The user query is normalized to fix typing mistakes. Product terms are expanded using a synonym dictionary. Candidate products are chosen using title and product text matching. Cosine similarity is then calculated between the query vector and candidate vectors.

The combined score implemented in the source code gives 50% weight to TF-IDF similarity, 30% to title-term coverage 10% to product-text term coverage and 10% to a stored recommendation score. Final sorting also considers rating and review count. Budget requests exclude products without a price and apply the extracted maximum price before ranking. This method is transparent and reproducible. It is not personalized and remains sensitive, to missing prices, duplicates and lexical mismatch.

$$score = 0.50s + 0.30t + 0.10x + 0.10r \quad (5)$$

where s is cosine similarity, t is title-term coverage, x is product-text term coverage, and r is the stored recommendation score. Rating and review count are used as secondary sorting criteria rather than as direct terms in the combined score.

3.6 Proposed System Architecture

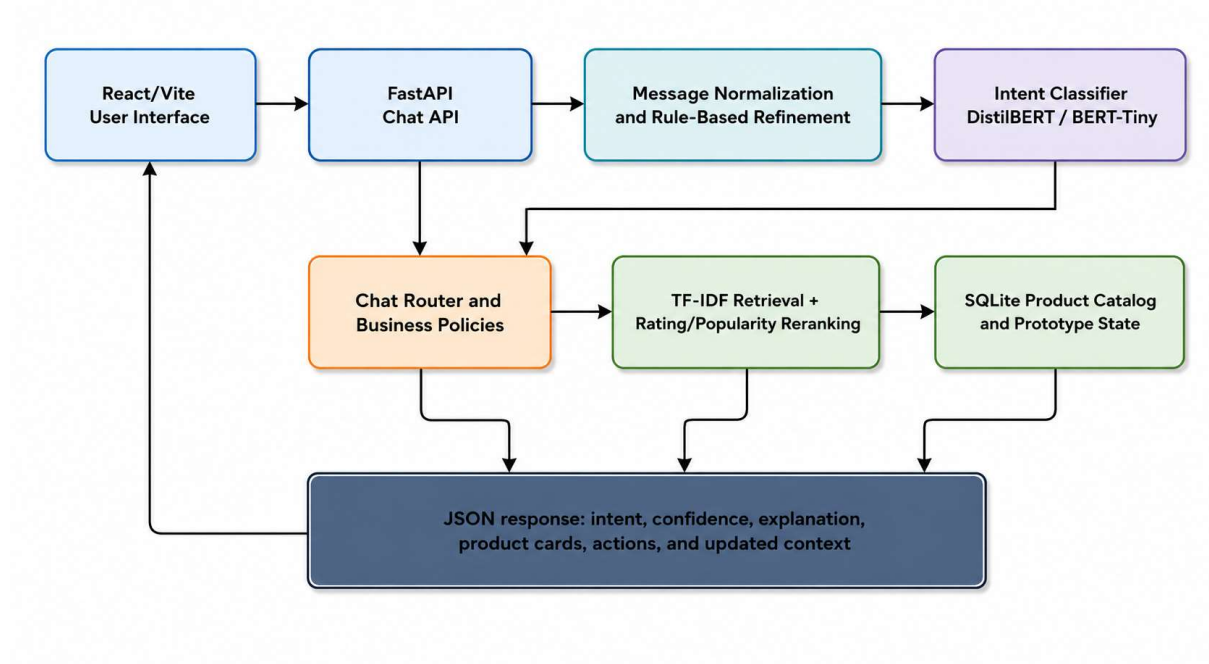


Figure 1 Proposed System Architecture

Table 8 components and responsibilities for store

Component	Responsibility
React/Vite UI	Collects messages, renders responses, product cards, model metadata, and user actions.
FastAPI Chat API	Receives requests, validates payloads, coordinates model inference, routing, retrieval, and response construction.
Normalization and Refinement	Normalizes misspellings, detects price and product expressions, and applies documented deterministic safeguards.
Intent Classifier	Produces the raw intent label and confidence using DistilBERT or BERT-Tiny.
Chat Router and Business Policies	Maps the final route to search, recommendation, product, cart, order, policy, support, or fallback behavior.
Recommendation Engine	Performs TF-IDF retrieval, lexical filtering, scoring, rating/popularity reranking, and price filtering.
SQLite Database	Stores product records and prototype tables for users, cart items, orders, chat logs, and recommendation logs.

Component	Responsibility
Response Formatter	Returns structured JSON containing intent, confidence, explanation, product cards, actions, and updated context.

3.6.1 UML Sequence Model

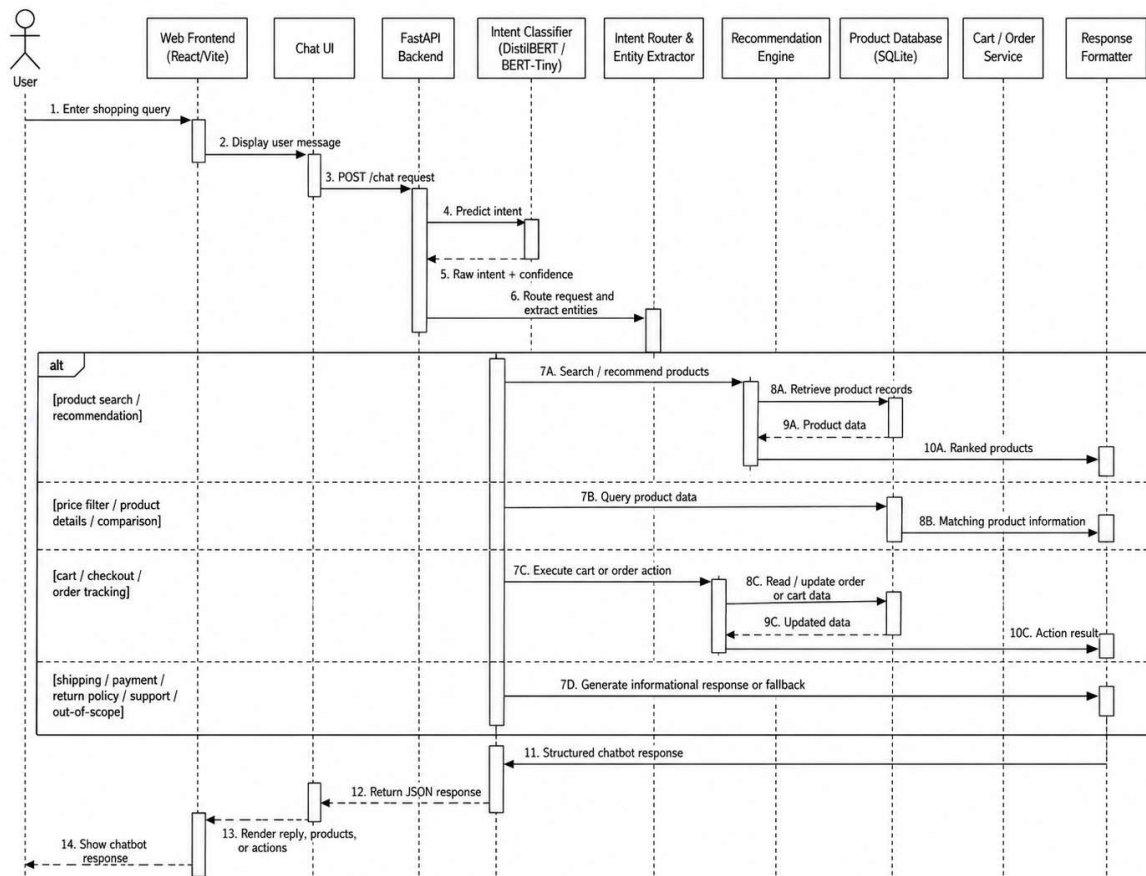


Figure 2 sequence diagram for store

The sequence model shows that a user message moves through the web frontend and chat interface before reaching the FastAPI backend. The classifier gives an intent and a confidence score. After that the router pulls out entities. Picks one of several application paths. Product-related requests go to retrieval and database services. Cart and order requests go to services. Informational and out-of-scope requests use a controlled response generation. The structured response is sent back to the frontend for rendering.

3.6.2 Use-Case Model

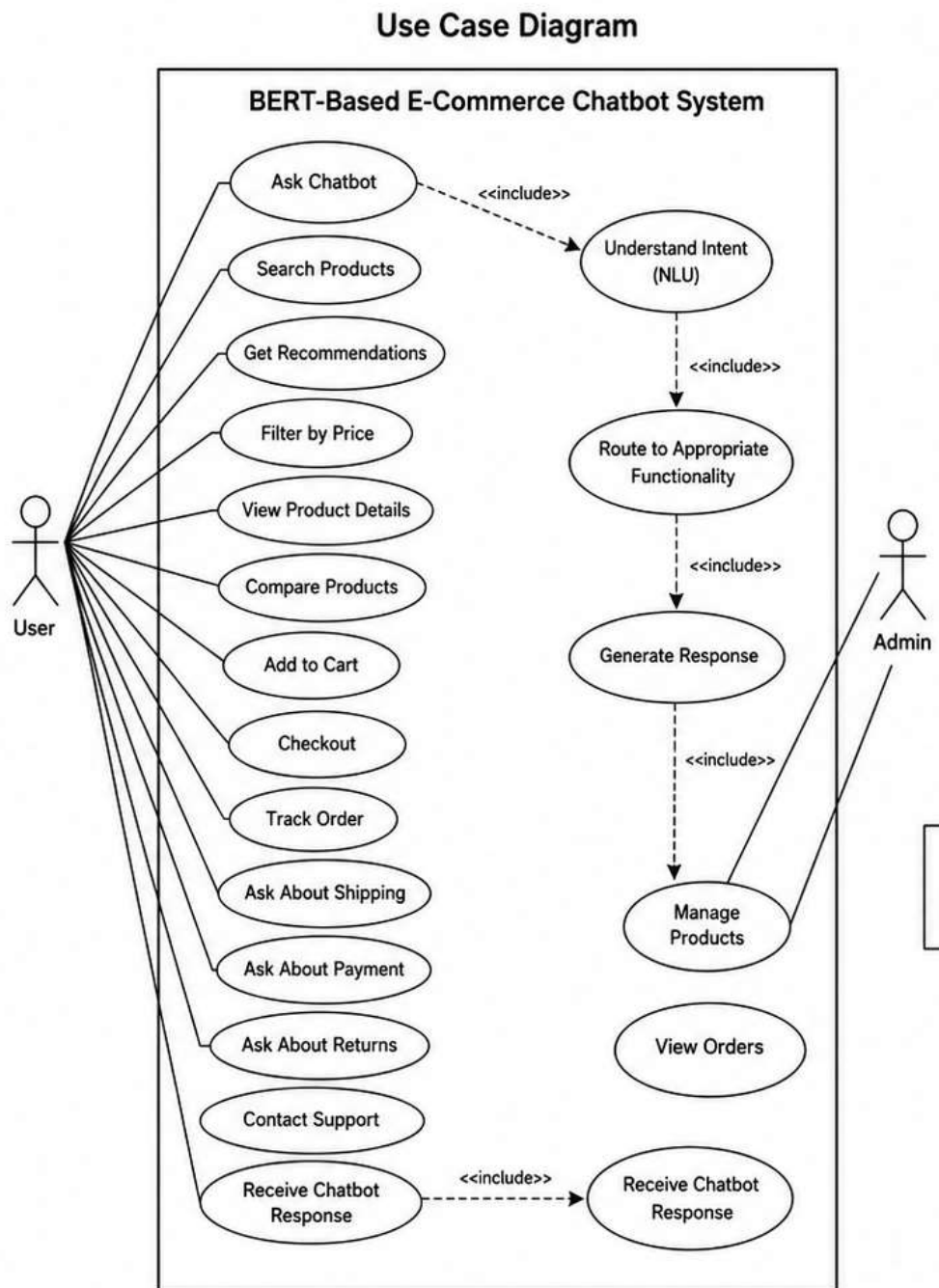


Figure 3 use-case diagram for store

3.6.3 Data-Flow Model

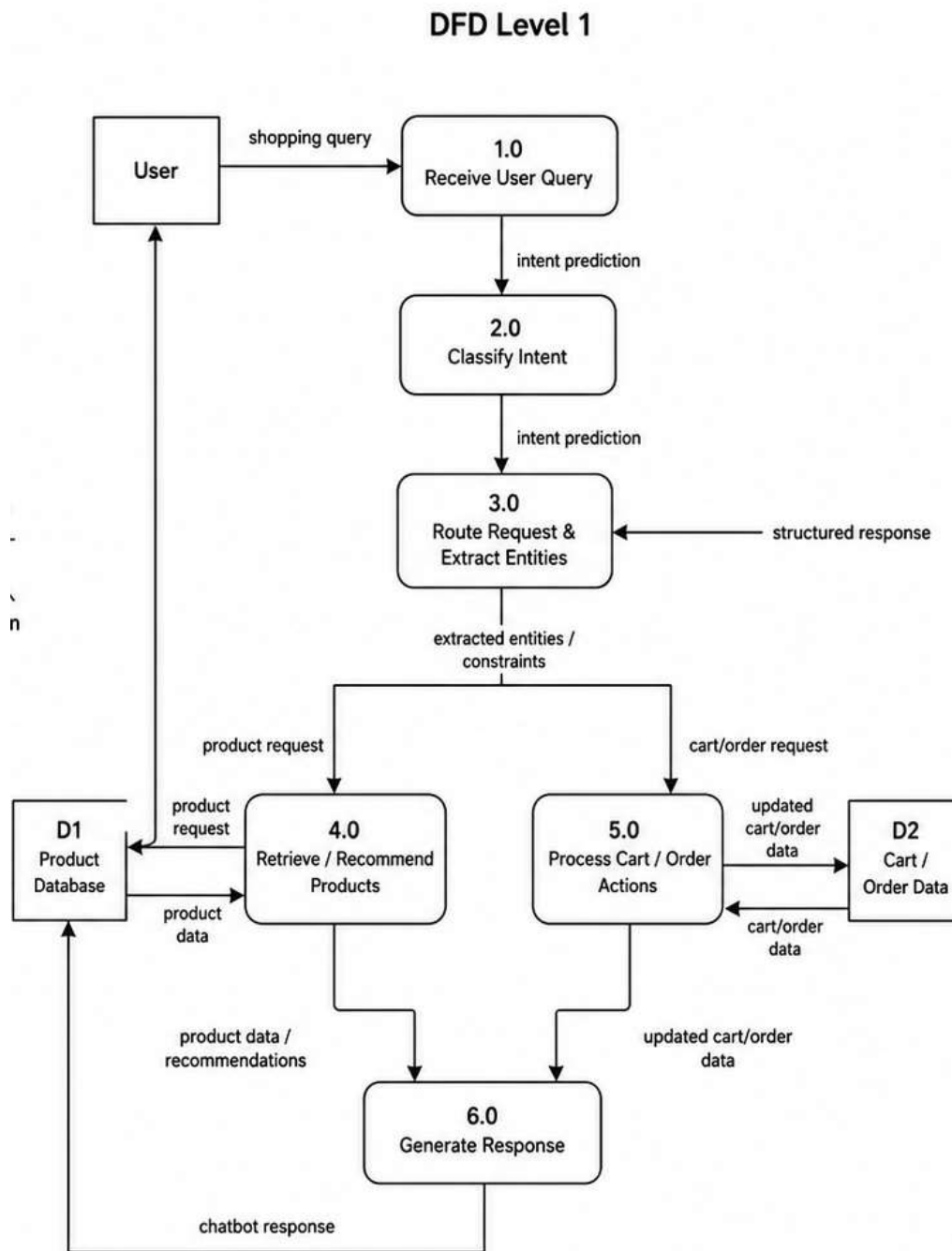


Figure 4 1 data-flow diagram level one

3.7 Software Implementation

3.7.1 Frontend

The frontend is built with React 19 and Vite 6. The frontend gives the home page, product catalog, chat interface, model-selection controls, cart interactions, login demo and research

dashboard. Product cards show category images that are assigned locally product titles, ratings, prices when available and actions for details, comparison and cart operations. The interface talks to the backend using JSON requests and shows the response returned by the chat endpoint.

3.7.2 Backend and API

The backend runs FastAPI. Sets up the SQLite database when the application starts. Cross-origin access is limited to Vite development addresses. Routers split product access, recommendations, chat operations and evaluation metrics. The chat route takes user input cleans the input and sends the input to the chosen model. Then the chat route applies routing rules calls the service. Returns a response. This separation makes it easier to inspect everything. You can test each endpoint on its own.

Table 9 Endpoints

Endpoint	Purpose
GET /api/health	Service availability check.
GET /api/products	Paginated catalog retrieval and filtering.
GET /api/products/categories	Product-category retrieval.
GET /api/products/{parent asin}	Single-product details.
POST /api/recommendations	Recommendation request through the retrieval service.
POST /api/chat	End-to-end chatbot request.
POST /api/chat/intent	Intent-only inference.
GET /api/chat/models	Available deployed classifiers.
GET /api/evaluation/intent	Primary intent metrics.
GET /api/evaluation/recommendation	Recommendation diagnostic metrics.
GET /api/evaluation/dataset-cleaning	Dataset-cleaning summary.

3.7.3 Database

The embedded SQLite database holds products, users, cart_items orders, chat_logs and recommendation_logs tables. The products table is the source for search and retrieval. The other tables help support the prototype state and simulated application behavior. At inspection time the products table had 8,000 records. The users, cart and orders tables were empty before any interaction. This shows that the project is a research prototype and not a real store.

3.7.4 Chat Routing and Business Logic

The routing layer keeps both the intent and the final route. Explicit rules find price expressions, product keywords, detail requests, checkout terms, shipping and payment words, tracking references and common typos. This design lets deterministic refinement happen if the model output clashes with an expression. Override reasons are returned so you can audit the process. This behavior is evaluated separately from classifier accuracy.

3.8 Evaluation Protocol

Table 10 Evaluation measures

Measure	Interpretation
Accuracy	Overall proportion of correctly classified test examples.
Weighted Precision/Recall/F1	Class metrics weighted by class support; useful for the observed distribution.
Macro Precision/Recall/F1	Unweighted mean across classes; highlights performance on smaller classes.
Confusion Matrix	Distribution of predicted labels for each true label.
Coverage@5	Whether five items were returned for each selected retrieval query; a system diagnostic.
Mean Similarity@5	Average TF-IDF similarity of the top five candidates; a lexical diagnostic.
Mean Reranked Score@5	Average combined retrieval score for the top five candidates.
Functional Completion	Whether the selected scenario returned the expected response type after the full pipeline.
Override Rate	Proportion of scenarios in which the final route differs from the raw intent.
Recorded Latency	Observed response time in the local sequential test environment; not a hardware-independent benchmark.

The controlled classifier experiment compares regression. Distilbert using the same held out examples. I report the BERT Tiny result. Recommendation evaluation uses six queries with k equal to 5 because there are no human relevance labels or user item interactions. Application verification uses twelve chosen chatbot scenarios and fifteen automated assertions. These layers answer research questions. They are not merged into one score.

3.9 Reproducibility with Ethics and Privacy

The archive holds source code, model configurations, trained weights, notebooks, metrics, screenshots and test reports. The primary configuration notes the model name, maximum sequence length, performance values and test set size. The BERT Tiny configuration notes its model name, data paths, epochs, batch size, sequence length learning rate and sampling cap. The Python requirements file lists dependency families but not versions, which limits

reproducibility. The frontend package file records package versions. The functional evaluation used messages defined by researchers and project data from customer conversations. A production system would need consent, data minimization, retention controls, access controls, secure logging and a clear policy on privacy and whether conversations can be reused for training. Payment, delivery and order responses, in the system are simulations. You should not read these as transactions.

3.10 Chapter Summary

This chapter recorded the implementation and experimental method. It described the datasets, cleaning and split choices controlled baseline, DistilBERT and BERT-Tiny configurations, product retrieval algorithm, architecture, UML models, frontend, backend, database, routing logic, evaluation measures and ethical limits. Chapter 4 shows the resulting evidence. Explains it in relation, to the research questions.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Dataset and Training Results

The cleaning procedure kept 79.9 % of the raw intent corpus. Product recommendation and out-of-scope were the groups while product_search and product_details were the smallest. Stratification kept each class represented the way in the training, validation and test sets. Figure (2) shows how the cleaned classes are spread. The imbalance was moderate not extreme. The difference in support is enough to report both weighted and macro F1. Figure (3) shows the learning behavior across four epochs. DistilBERT's training loss decreased throughout optimization, while validation loss declined from 0.515 after the first epoch to 0.183 after the fourth. Validation accuracy increased from 0.908 to 0.958 and macro F1 from 0.912 to 0.960. The simultaneous improvement of validation metrics and reduction in validation loss did not indicate late-epoch divergence during the four-epoch run.



Figure 5 Distribution of the 20 intent classes after primary data cleaning.

The class distribution is moderately imbalanced. Product recommendation and out-of-scope contain comparatively large numbers of examples, while product_search and product_details are smaller. Stratified splitting preserves these proportions across training,

validation, and test sets. Weighted and macro metrics are therefore reported together so that overall performance is not interpreted without considering smaller classes.

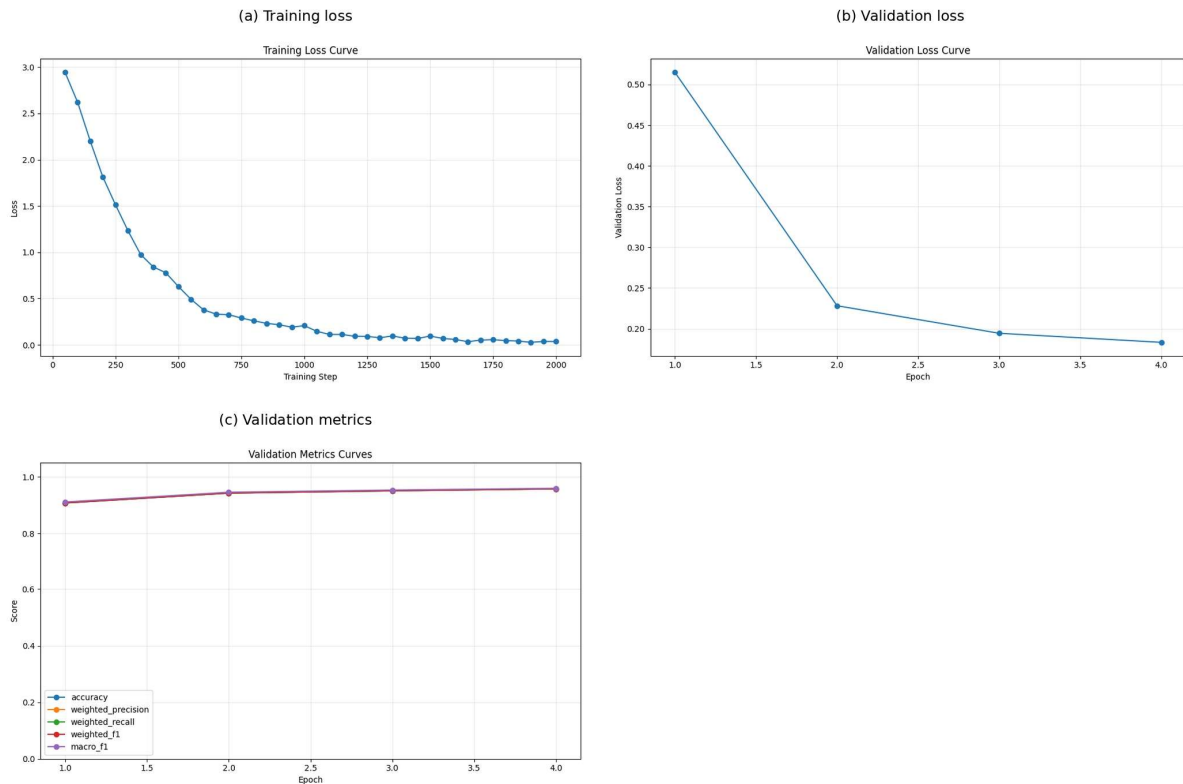


Figure 6 DistilBERT training and validation behavior across four epochs.

Training loss decreased throughout optimization, while validation loss fell from approximately 0.515 after the first epoch to 0.183 after the fourth. Validation accuracy increased from approximately 0.908 to 0.958, and macro F1 increased from approximately 0.912 to 0.960. The simultaneous improvement in validation metrics and reduction in validation loss does not indicate late-epoch divergence during the four-epoch run. However, one fixed seed and one split do not establish stability across repeated trials.

4.2 Controlled Classification Results

On the 1,738-sample main test set the TF-IDF logistic-regression baseline scored 0.8993 accuracy 0.8978 weighted F1 and 0.9021 macro F1. DistilBERT scored 0.9563 accuracy, 0.9564 F1 and 0.9595 macro F1. DistilBERT's weighted-F1 improvement was 0.0585. Because both models used the held-out examples the comparison shows a controlled difference for this data. However this result comes from one fixed seed and lacks confidence intervals, repeated-run tests or external validation; it should be read as descriptive evidence for the dataset.

Table 11 classification performance for test

Model	Accuracy	Weighted precision	Weighted recall	Weighted F1	Macro F1
TF-IDF + logistic regression	0.8993	0.8992	0.8993	0.8978	0.9021
DistilBERT	0.9563	0.9572	0.9563	0.9564	0.9595
Absolute difference	+0.0570	+0.0580	+0.0570	+0.0585	+0.0573

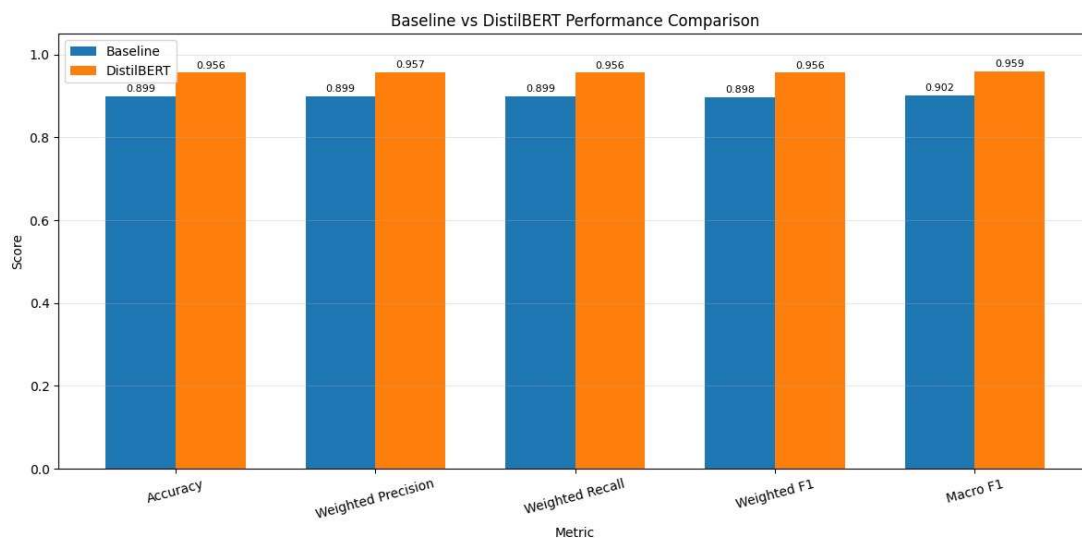


Figure 7 Controlled comparison between TF-IDF logistic regression and DistilBERT.

DistilBERT improves weighted F1 by 0.0585 and accuracy by 0.0570 compared with the controlled baseline. Because both models use the held-out examples the result shows that contextual representations add value for this data. The conclusion is limited to one fixed-seed run. Has no confidence intervals or repeated-run tests.

4.3 Per-Class Analysis

DistilBERT obtained an F1-score of 1.00 for compare_products, price_filter, product_details, product_recommendation, and product_search. The lowest F1-scores were out_of_scope (0.892), place_order (0.912), issue_support (0.913), order_tracking (0.918), and account_management (0.928). The lower scores appear in categories whose words overlap with support, payment order-management or product-inquiry language. The normalized confusion matrix shows that most points lie on the diagonal. The remaining mistakes cluster in semantically related actions instead of being spread out.

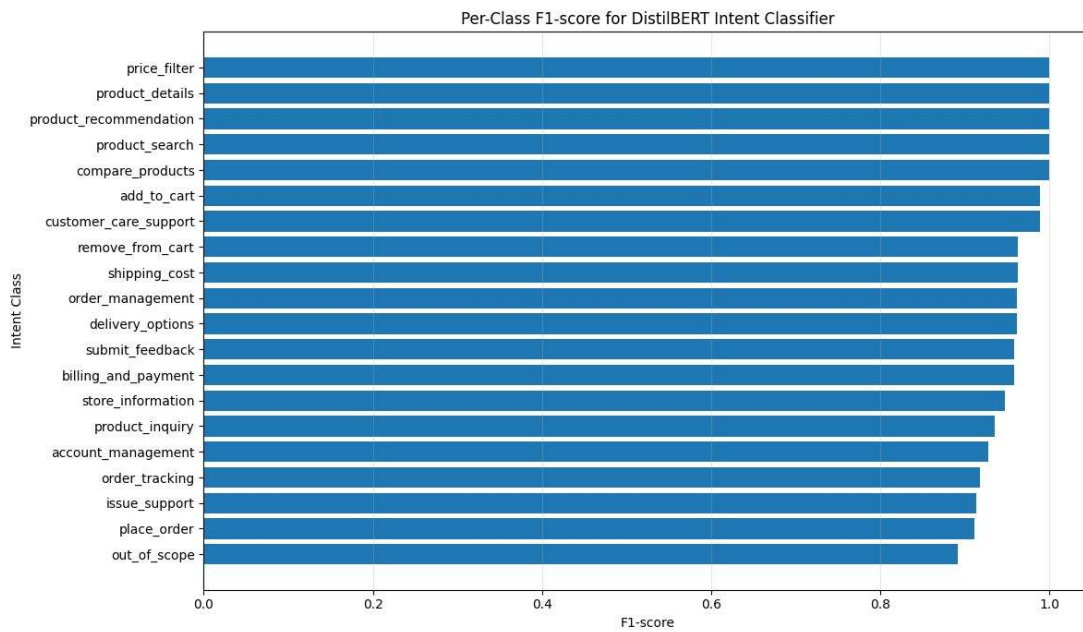


Figure 8 DistilBERT F1-scores

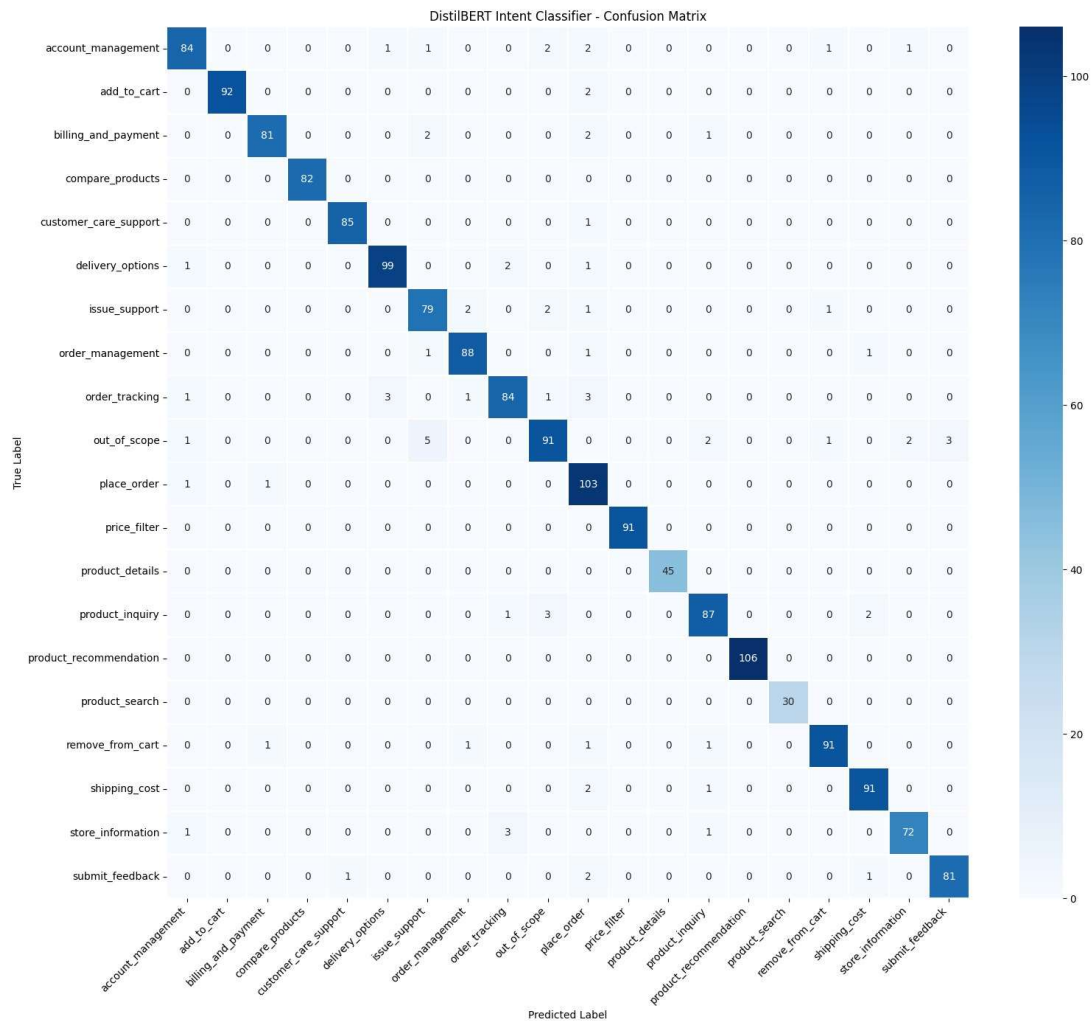


Figure 9 DistilBERT confusion matrix on test.

Table 12 results of model

Intent	Precision	Recall	F1	Interpretation
out_of_scope	0.919	0.867	0.892	Lowest F1; unsupported requests remain difficult
place_order	0.851	0.981	0.912	High recall with lower precision; overlaps service/payment language
issue_support	0.898	0.929	0.913	Confusion with general support-related intents
order_tracking	0.933	0.903	0.918	Overlap with order management and service requests
account_management	0.944	0.913	0.928	Some ambiguity among account and support requests
price_filter	1.000	1.000	1.000	Highly distinctive price expressions
product_recommendation	1.000	1.000	1.000	Distinctive recommendation phrasing
compare_products	1.000	1.000	1.000	Explicit comparison vocabulary

The strongest categories use distinctive lexical patterns, including explicit comparison phrases, recommendation language, product-search wording, and numerical price constraints. Lower-performing categories overlap with support, account, payment and order language. The out_of_scope category is still hard because unsupported requests are open-ended and can look like store messages. These observations suggest that future work should refine the intent taxonomy collect negative examples and add clarification steps for unclear predictions.

4.4 Auxiliary BERT-Tiny Results

The auxiliary BERT-Tiny model scored 0.8548 accuracy, 0.8578 weighted precision, 0.8548 weighted recall, 0.8466 weighted F1 and 0.8563 macro F1 on its 1,928-sample test split. Training accuracy rose early while validation performance peaked before the final epochs and validation loss grew later showing that the chosen ten-epoch setting did not give the generalization as the main DistilBERT run. The result shows that the model is deployable and fast not that its architecture is weaker in controlled tests. A fair comparison would need cleaning, splits, sampling, optimisation search and repeated runs.

Table 13 BERT-Tiny results

Metric	Value	Qualification
Test loss	0.7399	Separate 1,928-item test split
Accuracy	0.8548	Not directly comparable with the primary experiment

Metric	Value	Qualification
Weighted precision	0.8578	Training was capped at 200 cases per intent
Weighted recall	0.8548	Different cleaning and split
Weighted F1	0.8466	Reported as auxiliary deployment evidence
Macro F1	0.8563	Twenty intent labels

BERT-Tiny achieved 0.8548 accuracy and 0.8466 weighted F1 on its separate 1,928-item test set. The model is small and deployable. Its result does not prove that the architecture is weaker than DistilBERT. A fair comparison would need cleaning, sampling, data splits, optimisation budgets and repeated runs.

4.5 Recommendation Diagnostics

The recommendation engine was checked with selected retrieval queries

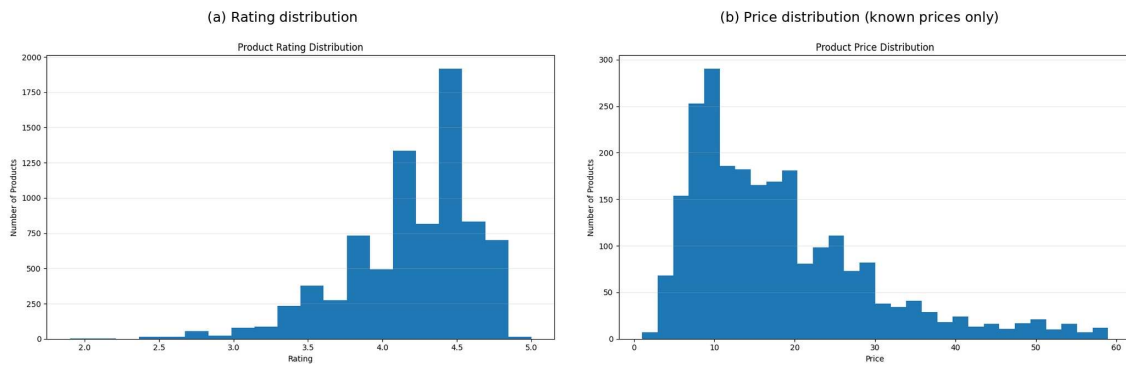


Figure 10 Rating and price distributions.

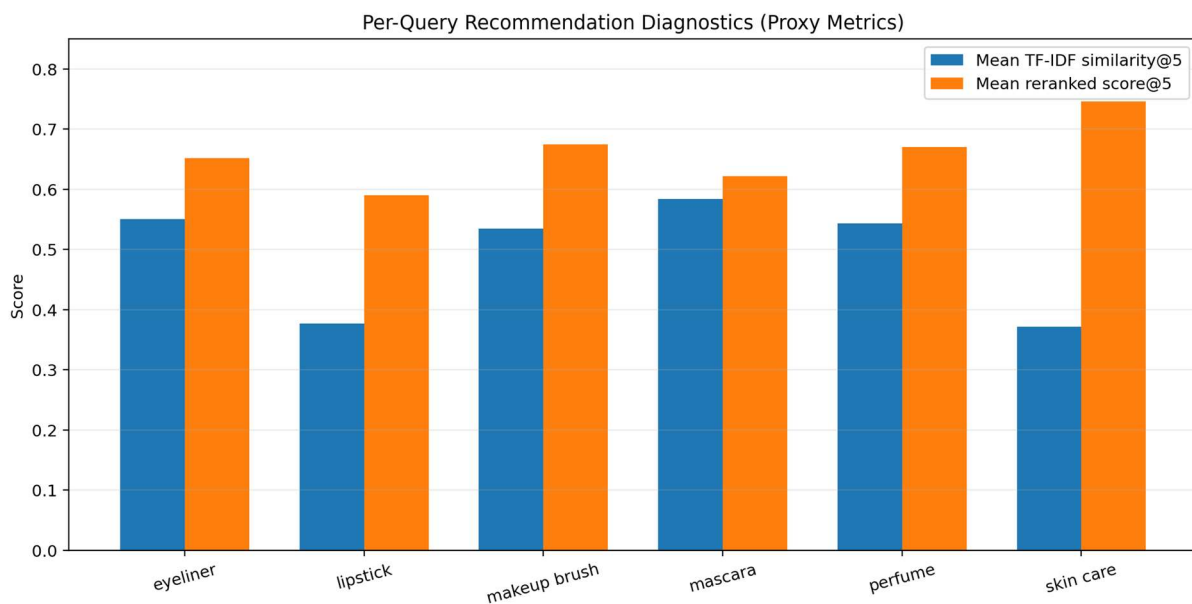


Figure 11 six representative queries proxy metrics

Table 14 Recommendation diagnostics

Query	Items returned	Mean similarity@5	Mean reranked score@5
eyeliner	5	0.5506	0.6516
lipstick	5	0.3768	0.5900
makeup brush	5	0.5343	0.6748
mascara	5	0.5840	0.6217
perfume	5	0.5432	0.6704
skin care	5	0.3713	0.7461
Overall	30	0.4934	0.6591

The engine returned five products for all six chosen queries giving coverage@5 of 1.0 by the project's definition. Mean similarity@5 was 0.4934. Mean reranked score@5 was 0.6591. Mascara had the average lexical similarity while lipstick and skin care had lower values. These results show that the service returns and ranks candidates but they do not measure user happiness or precision against human relevance labels.

Qualitative inspection showed limits. Lexical similarity can return accessories or brushes that share words with the requested product duplicate titles may. Related products may be missed when the query and product description use different terms. Missing price data also reduces budget-request coverage. Semantic embeddings, product-type classifiers, duplicate removal and human relevance checks are needed for recommendation evaluation.

4.6 Chatbot and API Experiments

4.6.1 Executed Chatbot Scenarios

DistilBERT's mean confidence across the chosen scenarios was 92.0 % compared with 82.0 % for BERT-Tiny. Median recorded latency was 0.066 seconds for DistilBERT. 0.011 Seconds for BERT-Tiny in the local test sequence. The latency figures include the project's runtime environment and model cache state; they are not benchmarks.

Product-returning scenarios produced 26 product cards per model across the set. Policy and support scenarios returned text without product cards matching the intended route design.

Table 15 Test part A

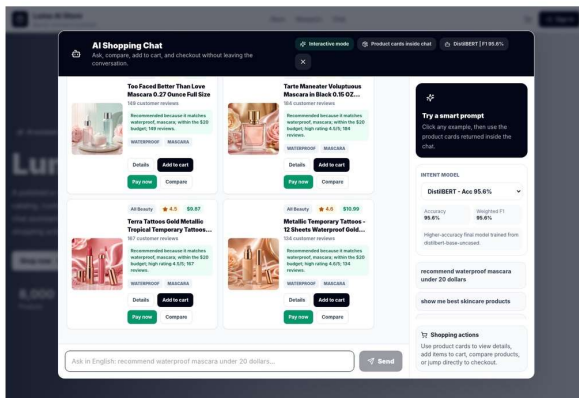
ID	User request	Raw intent → final route	Confidence	Observed output
CT-01	Waterproof mascara under \$20	price_filter	98.6%	6 product cards
CT-02	Niacinamide serum for dry skin	product_search	47.2%	6 product cards

ID	User request	Raw intent → final route	Confidence	Observed output
CT-03	Compare two eyeliners	compare_products	98.8%	2 product cards
CT-04	Shipping cost	shipping_cost	99.2%	Policy/support response
CT-05	Track ORD-123456	order_tracking	99.1%	Policy/support response
CT-06	Damaged product support	issue_support	99.1%	Policy/support response

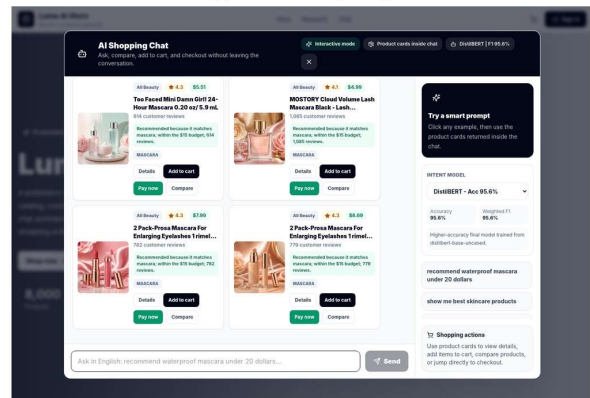
Table 16 Test part B

ID	User request	Raw intent → final route	Confidence	Observed output
CT-07	Visa payment	billing_and_payment	94.4%	Policy/support response
CT-08	Football joke	out_of_scope	99.5%	Policy/support response
CT-09	Misspelled mascara under \$15	remove_from_cart → price_filter	80.5%	6 product cards
CT-10	Oral-B toothbrushes	product_inquiry	90.7%	6 product cards
CT-11	Checkout request	billing_and_payment → place_order	97.7%	Policy/support response
CT-12	Password reset	account_management	99.2%	Policy/support response

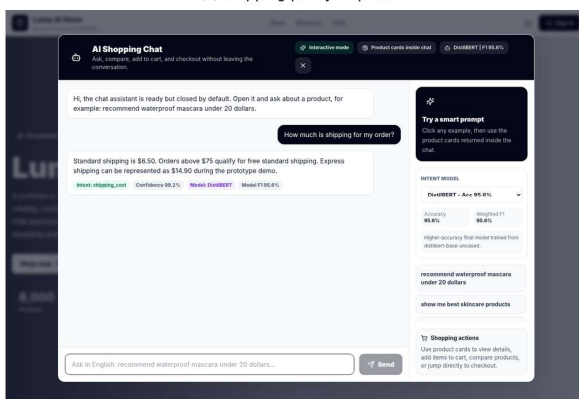
(a) Price-filtered recommendation



(b) Typo-tolerant budget request



(c) Shipping policy request



(d) Out-of-scope request

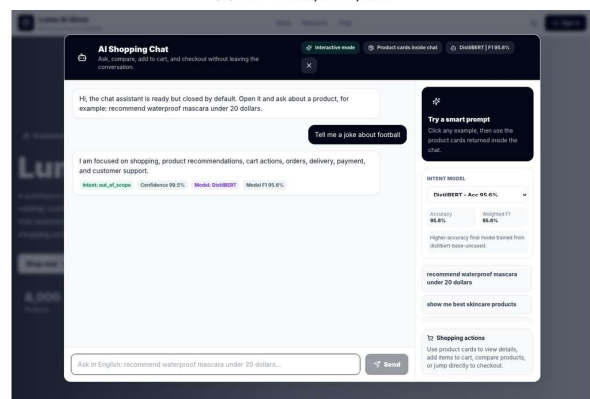


Figure 12 Executed chatbot tests for product, typo, policy, and out-of-scope requests.

For each deployed model the raw classifier intent matched the application route in ten of twelve cases (83.3 %). The two changed cases were the typo-heavy budget request and the

checkout request. DistilBERT predicted `remove_from_cart` for the mascara message before the explicit budget rule selected `price_filter`. It predicted `billing_and_payment` for "I want to checkout" before the checkout rule selected `place_order`. These cases show the value of deterministic refinement and the importance of reporting the raw prediction rather than blaming the classifier alone for scenario success.

The chosen DistilBERT scenarios recorded a confidence of about 92.0 % compared with about 82.0 % for BERT-Tiny. Median local latency was 0.066 seconds for DistilBERT and about 0.011 seconds for BERT-Tiny. These values were collected sequentially in one runtime, with response paths and cache states; they are not hardware-independent benchmarks.

4.6.2 Automated API Verification

Table 17 Test application verification

Test group	Representative checks	Result	Interpretation
Availability and data	Health, model list, catalog and category retrieval	Passed	Core services and local data were accessible
Intelligent functions	Search, recommendation, price filtering, comparison	Passed	Expected response structures and product cards were returned
Business scenarios	Shipping, payment, tracking, support, out-of-scope	Passed	Expected policy or support routes were returned
Total	15 automated assertions	15/15 passed	Functional verification only; not production certification

The final verification evidence has two layers. The documented regression script ran 52 chatbot cases covering requests, typographical errors, context-dependent actions result limits, product details, comparison, cart behavior, shipping, payment, tracking, support and out-of-scope inputs; all 52 cases passed. A second focused validation suite ran 15 checks covering service health the 8,000-record catalog, product categories, loading of DistilBERT and BERT-Tiny, budget-constrained recommendation, product-type relevance, comparison uniqueness, typographical-error recovery, shipping responses, negative budgets, ambiguous requests, out-of-catalog requests, blank messages, unsupported model

4.6.3 Interface Verification

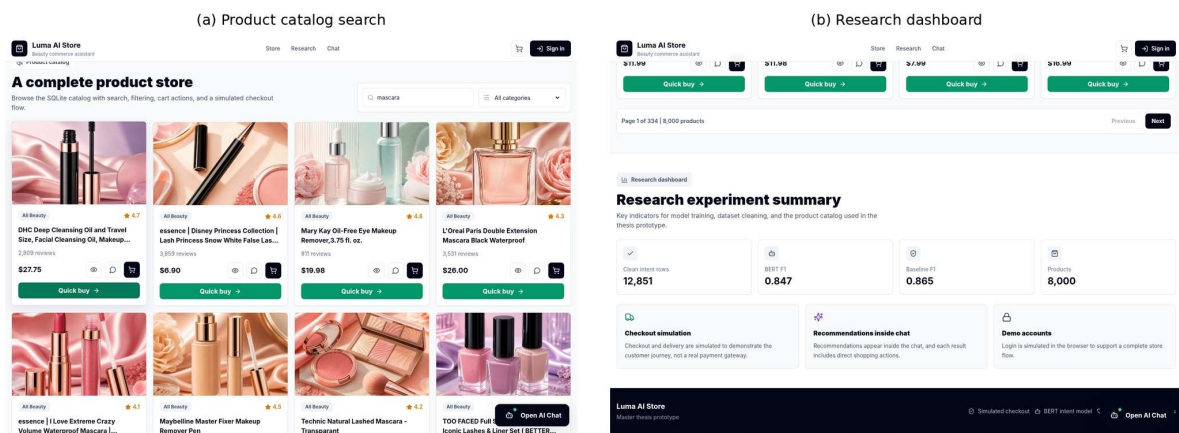


Figure 13 home, catalog-search, product interfaces.

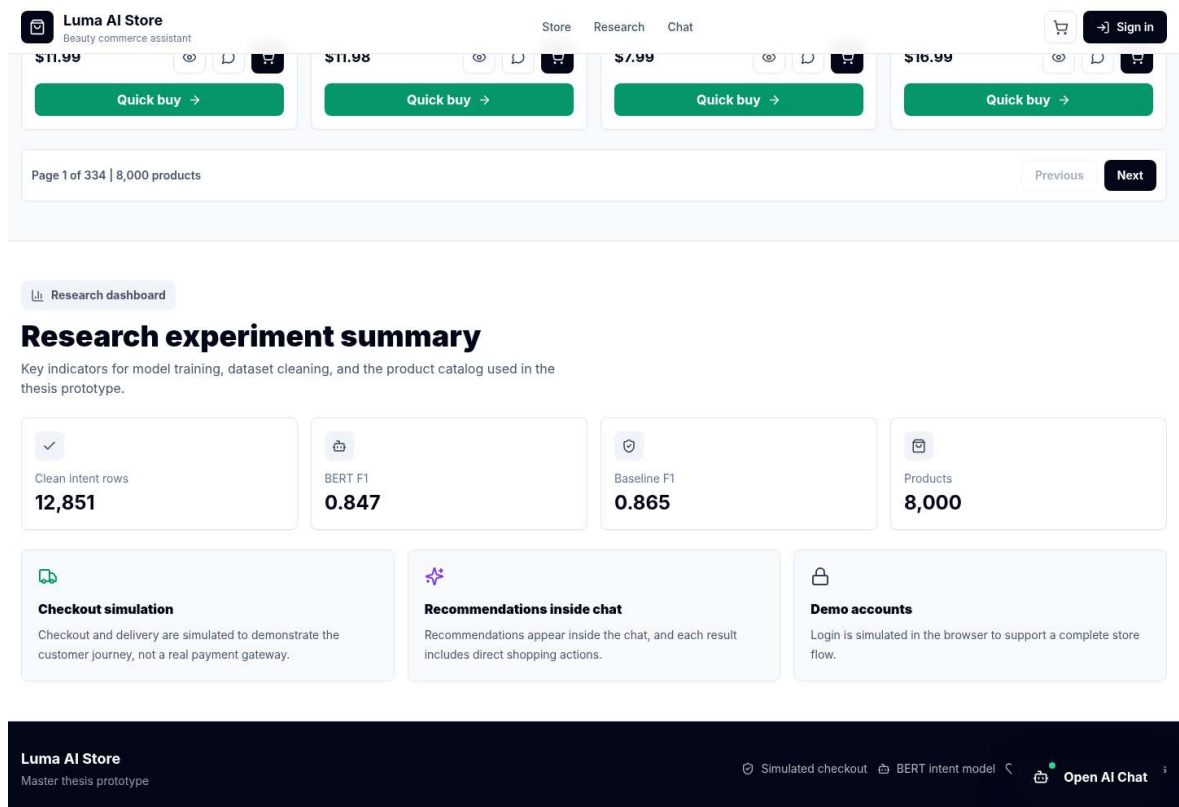


Figure 14 research summary screen

The interface screenshots show that users can reach the landing page look through the catalog open the chat window see the product cards that come back and check research metrics. The dashboard shows model data and evaluation files so the project can be Tested. Visual proof. It does not replace real function tests or model numbers.

4.6.4 Final Runtime Verification and Hidden-Issue Corrections

I ran the project archive again from its packaged backend using the trained DistilBERT and BERT Tiny models, the SQLite catalog, local product assets and the frontend bundle. The final visual revision restored the approved identity that was seen in the project: a navy

navigation system paired with cream, peach, soft rose and warm gold product accents. The hero image no longer relies on an internet URL; it uses a packaged beauty product asset. The review looked at failures. Hidden logical defects that could show up during a fresh start, such as invalid input, recommendation ranking, product comparison, typographical errors missing prices or requests outside the beauty catalog. I accepted a correction after the problematic behavior was re-executed and added to the 52-case regression suite or the focused 15-case runtime audit.

Table 18 implementation of requirements

Detected issue	Correction implemented	Verification evidence
Hero depended on an external image URL and did not reliably display offline.	Replaced the external dependency with a packaged beauty-product hero asset and referenced it from the source and built frontend.	The approved-identity homepage loaded with the local hero while the environment was offline.
The runtime failed when Transformers was unavailable.	Added a project-local PyTorch and safetensors inference runtime for DistilBERT and BERT-Tiny.	Both models returned valid predictions; runtime backend reported project-local-pytorch.
Database-reading services could be imported before schema initialization.	Initialized the SQLite schema before router/service imports and added an empty-catalog guard.	Fresh startup and health/catalog checks passed.
Chat and recommendation payloads accepted invalid or unbounded values.	Added whitespace, length, model-key, and recommendation-limit validation.	Invalid requests returned HTTP 422.
Catalog LIKE queries treated % and _ as wildcards.	Escaped user search patterns and added query-length limits.	Literal wildcard search no longer returned the complete catalog.
A declared evaluation route referred to a missing artifact.	Returned the packaged evaluation summary with an explicit CPU-runtime note when no separate file exists.	All evaluation endpoints returned HTTP 200.
Raw chat logs could retain e-mail addresses and order identifiers.	Added targeted redaction before writing research logs.	Stored log replaced identifiers with [email] and [order-id].
Price-filter title preference treated an attribute as a product type.	Removed waterproof from product-type matching while retaining it as a query attribute.	Returned price-filter titles were mascara/lash products within the requested budget.
The database utility imported an excluded data-engineering module.	Replaced it with a packaged-database validation utility.	Utility reported 8,000 available products without import errors.
Development API configuration and preview CORS were too narrow.	Added an environment-based API base and allowed Vite development and preview origins.	Source inspection and frontend execution confirmed local API access.
Ranked recommendations could display duplicate normalized titles.	Added post-ranking visible-title deduplication while preserving the strongest-ranked record.	The eyeliner validation returned six results with six unique visible titles.
Negative and zero budgets were accepted or interpreted inconsistently.	Added explicit positive-price validation before retrieval.	Requests below -5 and 0 returned controlled validation messages and no products.
Textual, malformed, and unrealistic	Added parsing for common written numbers and	“twenty” was parsed as 20; \$abc and

budgets were not handled consistently.	rejection for malformed or values above \$10,000.	999999 were rejected with specific guidance.
The frontend generated deterministic estimated prices when catalog prices were missing.	Removed the estimated-price fallback, displayed “Price unavailable,” and blocked cart/checkout actions without a verified price.	Source and built bundle contain the corrected behavior and no “Est.” price fallback.
Generic API failures exposed raw response text rather than a useful validation message.	Parsed FastAPI detail fields and preserved a separate connection-unavailable message.	Error-state screenshots show understandable validation and recovery messages.
A descriptive attribute could dominate lexical retrieval even when it did not identify the requested product type.	Separated product-type anchors from descriptive attributes and retained rating/popularity reranking.	The waterproof-mascara test returned mascara products only and excluded unrelated waterproof items.
The final built frontend did not initially include the last source-level price-integrity correction because package download failed.	Patched the supplied prebuilt bundle to match the verified source behavior and retained the source as the rebuild reference.	The final bundle contains “Price unavailable,” blocks unpriced cart additions, and passes the asset/bundle validation checks.
The project did not preserve a single human-readable map of success and error screenshots.	Added qa_screenshots/SCREENSHOT_MANIFEST.md and grouped evidence by successful execution and validation/recovery states.	The final archive contains eight success images and eight error/recovery images.
The redesigned interface did not match the visual identity shown in the original product screenshots.	Restored the approved navy, cream, peach, soft-rose, and warm-gold identity and aligned buttons, cards, navigation, and chat surfaces.	The final homepage screenshot documents the approved identity in the running application.
Product-card images could be unrelated to the visible product title when chosen by card position.	Mapped local images according to normalized product type and title rather than display order.	Executed recommendation and comparison screenshots show category-consistent beauty-product imagery.

Missing prices stay a declared data limitation. **The corrected interface now shows a price.** Stops cart or checkout actions for items that do not have a verified price rather than inventing an estimated amount. The final recommendation logic also locks a product type anchor so that a word, like waterproof cannot push product types above the requested mascara category.

The following screenshots were taken from the running application after the approved visual identity, local hero asset, recommendation logic, duplicate title removal, product type anchoring and validation messages were implemented. I showed the images at a width that lets you see interface labels returned product cards and error messages clearly.

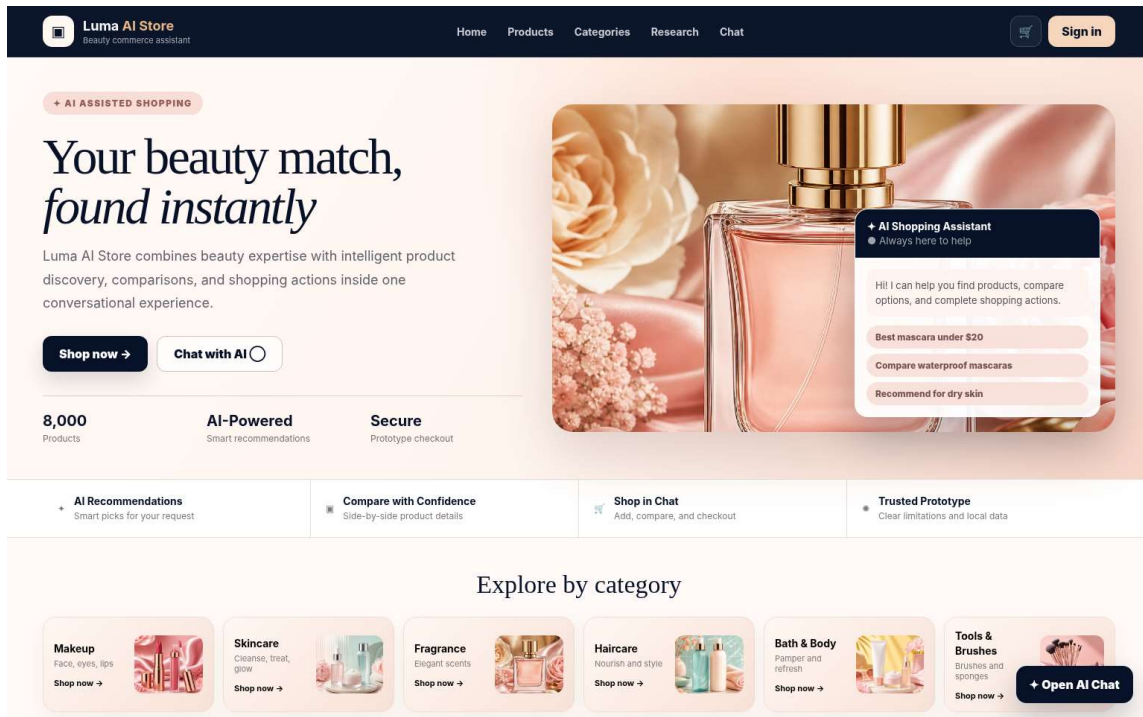


Figure 15 homepage

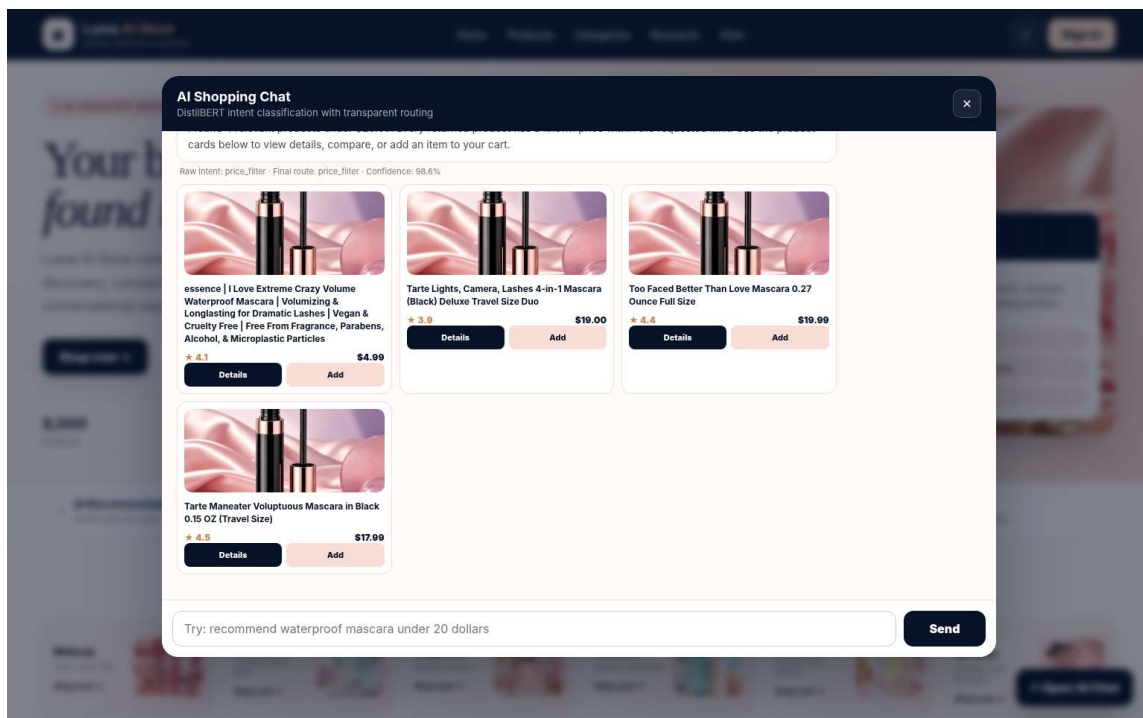


Figure 16 chat

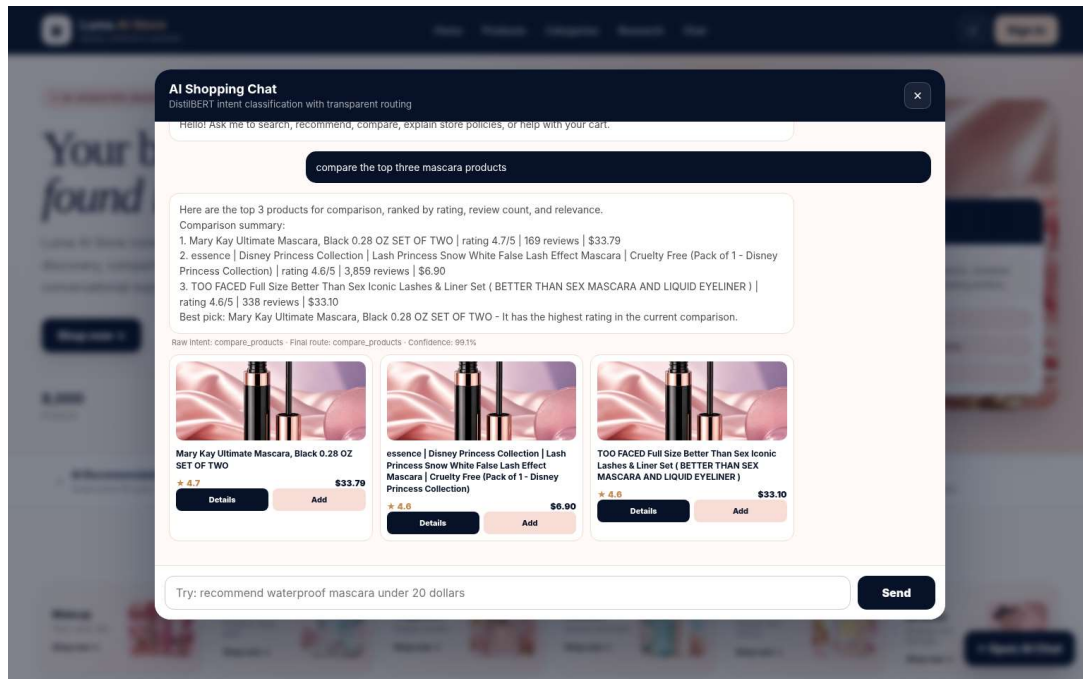


Figure 17 product comparison in chat

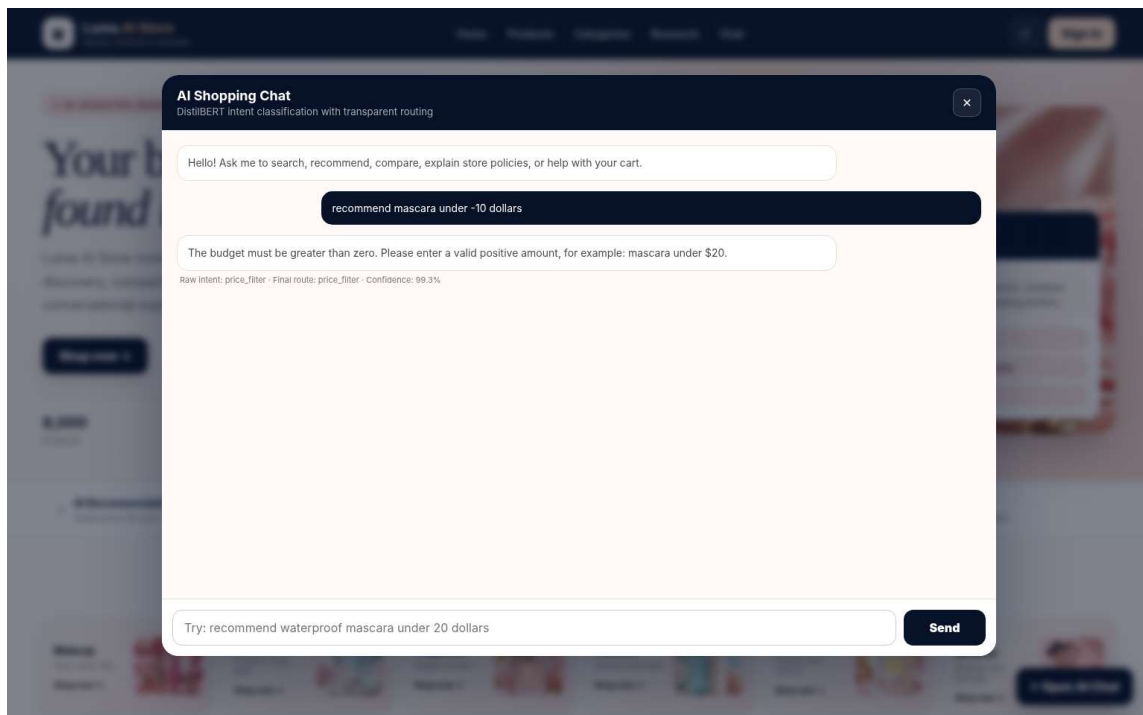


Figure 18 negative budget validation message

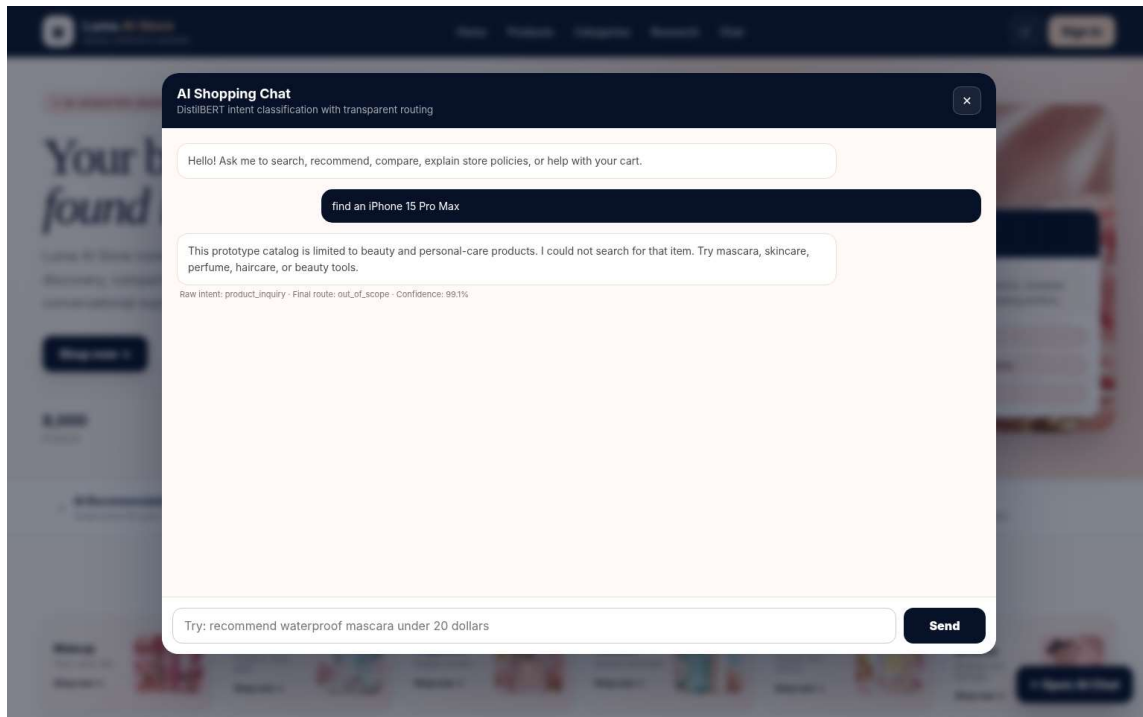


Figure 19 out of catalog response

I notice that Figures (15)–(20) show the difference between actions and controlled validation and recovery.

I see that Figures (15)–(18) document the approved interface look the price limited recommendation, comparison and typographical error recovery.

I see that Figures (19)–(20) show the expected replies to a budget and a product request that is, outside the catalog.

I understand that these error states are not crashes that cannot be fixed; they show that invalid or unsupported inputs are stopped before products or unsafe shopping actions become possible.

4.7 Discussion of Research Questions

Table 19 question findings

Research Question	Answer Supported by the Results
RQ1	DistilBERT achieved 0.9563 accuracy and 0.9564 weighted F1, exceeding the controlled TF-IDF baseline by 0.0570 accuracy and 0.0585 weighted F1.
RQ2	The most difficult classes included out_of_scope, place_order, issue_support, order_tracking, and account_management; errors concentrate among semantically related service and transactional intents.
RQ3	The deployed system uses a path in which the model gives a raw intent and confidence while deterministic safeguards, extracted constraints and business policies choose the final route.

Research Question	Answer Supported by the Results
RQ4	The twelve scenarios and fifteen automated checks show some coverage. Two scenarios needed routing overrides proving that functional success does not count as classifier accuracy.
RQ5	The recommendation engine gives ranked candidates. The evidence is only from proxy diagnostics. Missing prices, no user histories and no human relevance judgments stop us from claiming personalization or user satisfaction.

The controlled results answer RQ1. As shown in Table 4 and Figure 4 DistilBERT greatly improved classification compared to the TF IDF logistic regression baseline on the held out test set. The weighted F1 increase of 0.0585 shows that representations added value beyond unigram and bigram statistics for this data set. The conclusion is limited to the supplied data and a single fixed seed experiment; external validity needs data sets and repeated trials.

RQ2 is answered by the class level analysis in Figure 5 and Table 5. Perfect or near scores happened for intents with expressions, including price filtering and explicit product actions. Lower scores happened for out_of_scope, place_order, issue_support, tracking and account management utterances where the words overlap with service requests. This pattern means that future improvement should focus on negative examples unclear utterances, clarification strategies and calibrated confidence thresholds.

RQ3 is about how prediction turns into action. Figure 1 shows that the architecture uses a pipeline: the model supplies a probability based intent while explicit normalization and routing rules protect high value business actions and common noisy expressions. Tables 8 and 9 record both intents. The approach is auditable because the API logs the prediction, final route and reason for any override; however the presence of overrides means that final application behavior cannot be used as a measure of classifier accuracy.

RQ4 is answered by the executed scenarios and API report. Tables (8) and (9) Figure (7) Table (10) and Figure (8) show that the chatbot returned the expected response type for product discovery, budget filtering, comparison policy questions order tracking, damaged-item support, payment, account management and out-of-scope messages. These results show coverage under the chosen test conditions but they are not a replacement for randomly sampled real conversations, human judgments, error-severity analysis, stress testing or long-term reliability measurement.

RQ5 concerns limits of recommendation evidence. Table (7) and Figure (6) show that the content-based engine consistently returned candidates and produced similarity and reranking scores. These diagnostics confirm retrieval behavior only. Without user histories human

relevance judgments or interaction outcomes they cannot prove personalization, recommendation usefulness, diversity or satisfaction. Missing price values also limit price-based selection to items that have documented prices.

The selected chatbot cases also reveal a design trade off. Deterministic overrides improved task completion for price requests and checkout language. They can hide classifier errors if only the final route is reported. Both values must therefore be kept for auditability. During deployment the override rate should be watched as an indicator: a rising rate may signal model drift changes, in user language insufficient training coverage or routing rules that have become too broad.

The observed latency values should be interpreted with caution. They were collected one after another in a runtime and include different response paths; they are not controlled throughput benchmarks. DistilBERT's mean confidence and slower median latency match its capacity compared to BERT Tiny but hardware independent conclusions would need warm up control, repeated measurements, identical batches, concurrency tests and confidence calibration.

4.8 Limitations and Threats to Validity

Internal validity is limited by the use of one fixed seed and one split in the controlled experiment. The observed difference between the baseline and DistilBERT is descriptive and requires repeated runs for confidence intervals and significance analysis. BERT-Tiny was trained through a different pipeline, so its result cannot support architectural ranking.

Construct validity is limited by the available evaluation evidence. Functional completion is based on purposefully selected scenarios rather than a random sample of user conversations. Recommendation metrics measure retrieval behavior rather than perceived relevance, diversity, usefulness, or satisfaction. Confidence values are model outputs and have not been calibrated as probabilities of correctness.

External validity is limited by the English-language intent corpus, the beauty-focused product catalog, undocumented intent-data provenance, missing product prices, and the absence of production traffic. The results should not be generalized to Arabic, multilingual commerce, other product domains, real payment systems, or high-load deployment without additional evidence.

Implementation validity was strengthened by adding bounded dependency ranges, an offline PyTorch and safetensors runtime for both packaged Transformer models, early

database-schema initialization, message and model-key validation, budget validation, escaped catalog search patterns, privacy-oriented log redaction, duplicate-title suppression, verified-price handling, an environment-configurable API base, and a local hero asset. The final release includes source code, a tested prebuilt frontend, automated reports, execution logs, and success/error screenshots. Remaining implementation limits include browser-local simulated authentication, simulated payment and shipping, and the absence of production-scale security, accessibility, concurrency, and long-duration testing.

4.9 Chapter Summary

This chapter showed that DistilBERT substantially outperformed the controlled TF-IDF baseline on the supplied intent dataset, while BERT-Tiny provided separate lightweight deployment evidence. It identified class-level difficulties, demonstrated the operation and limitations of the content-based retrieval engine, and verified selected chatbot and API functions. The discussion answered the research questions while preserving the distinction between model accuracy, routing behavior, retrieval diagnostics, and software execution.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

Overall this study shows a research implementation. The best evidence comes from the controlled comparison of DistilBERT and the clear connection between classification and application actions. The main problems are that the intent corpus has no origin there are no repeated model runs the test set is small there are no user-item relevance labels. There is no testing with real users or in production. Admitting these limitations provides a foundation for future work rather than combining different results into one performance claim.

Thus the research answers the question with a conclusion: a small Transformer can do a good job at understanding intent within a working e-commerce application but reliable behavior comes from combining classification clear routing rules, product retrieval, database actions and user feedback. The main evidence shows that DistilBERT performs better than the controlled TF-IDF baseline on the data provided. It does not show that DistilBERT is better than BERT-Tiny because that model was trained in a way. The recommendation part is based on content not on personal information because the project has no user-item history or human evaluations; because of that the study does not show how good the recommendations are for real users how satisfied they are or if the system is ready for use.

The project meets the goal by connecting a trained intent classifier to a working e-commerce prototype. A user can submit a natural-language request, the backend can produce and record a raw intent, routing logic can refine the decision when explicit operational evidence is present, the appropriate product or business service can execute, and the frontend can render a structured result. The full path is supported by source code, trained models, a local database, metrics, automated tests, and screenshots.

The strongest empirical result is the controlled comparison between DistilBERT and the TF-IDF logistic-regression baseline. DistilBERT achieved 0.9564 weighted F1 compared with 0.8978 for the baseline on the same 1,738 held-out examples. The auxiliary BERT-Tiny result demonstrates that a smaller model can be deployed, but it does not establish an architectural comparison because the data pipeline differs. The recommendation engine remains a content-

based retrieval prototype, and its diagnostics do not establish personalized recommendation quality or user satisfaction.

5.2 Summary of Contributions

- A controlled comparison between TF-IDF logistic regression and DistilBERT on the same cleaned dataset and the same held-out test split.
- A transparent distinction between the primary DistilBERT experiment and the separate BERT-Tiny pipeline.
- An implemented full-stack e-commerce chatbot that connects intent prediction with search, recommendation, policy, cart, order, and support functions.
- An auditable distinction between raw model intent and final application route, including deterministic override reasons.
- A three-layer plan for checking the work that includes model metrics checking the recommendations. Running software scenarios.
- A clear analysis of the problems that includes the origin of the data how complete the catalog's no human evaluations and no testing in real situations.

Another practical contribution is the idea to monitor the override rate while the system is in use. If the override rate rises it could indicate that people are using words that the model does not handle well for questions or that the rules are correcting errors. Examining the final route, confidence, retrieval failures and user changes separately would provide a clearer view than relying on a single overall success metric.

5.3 Recommendations

5.3.1 Data

Make a data card that explains where the intent data came from how it was labeled, what the labels mean, how well the people who labeled it agreed, the license, consent and privacy checks. Gather conversations that have permission and are anonymized, especially looking for spelling mistakes different ways to say the same thing, follow-up questions and requests that are not supported. Improve how products are named, finding duplicates, category labels and prices.

5.3.2 Modeling

Do the test again with different random choices and share the average performance, standard deviation, confidence ranges and look at errors carefully. Train DistilBERT and

BERT-Tiny using the steps splits, number of samples and time for tuning before comparing them. Add ways to check how sure the model is and ask people to clarify when it's not sure.

5.3.3 Recommendation

Get evaluations of how relevant the recommendations are and check how good the rankings are, how diverse they are, how new they are, how helpful they are and how bad the failures are. Try using sentence and product embeddings or mixing ways to get better results. Only check - turn conversations after there is enough data to work with.

5.3.4 Deployment

Do tests for usability, accessibility, security, privacy handling a lot of users working at the same time recovering from problems and working for a long time before using it publicly. Put in place ways to keep users safe manage secrets keep records limit how much users can do watch the system back up data and handle problems. Make sure payment and shipping parts are real before using them for money.

5.3.5 Multilingual Extension

Add support for Arabic and other languages after getting enough labeled data and setting up how to handle each language how to split words how many dialects to cover, how to test code mixing and how to check with people. Arabic support should not be assumed just because there is an abstract because the current system and tests are only for English

5.4 Future Work Roadmap

Table 20 Future-work points

Horizon	Area	Planned Work
Short term	Reproducibility	Publish a locked environment, data card, model card, and versioned test suite.
Short term	Data quality	Complete product prices, normalize titles, remove duplicates, and validate categories.
Medium term	Model robustness	Run repeated experiments, calibrate confidence, and add clarification for ambiguous requests.
Medium term	Recommendation quality	Collect relevance judgments and compare lexical, semantic, and hybrid retrieval.
Medium term	User evaluation	Conduct task-based usability and satisfaction studies with consented participants.
Long term	Production engineering	Implement secure identity, real integrations, monitoring, load testing, recovery, and governance.
Long term	Multilingual capability	Collect Arabic and code-mixed data and evaluate language-specific

Horizon	Area	Planned Work
		models and taxonomies.

5.5 Final Statement

This study shows that a small Transformer can understand intent, within a functioning system but reliable behavior depends on combining classification, clear routing, product retrieval, database services and user feedback. The value of the work is. The system that was built and the clear way of separating the evidence. This separation gives a base for future work without saying more, than what is proven.

References

- Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6), 734–749. <https://doi.org/10.1109/TKDE.2005.99>
- Aly, W. M., Eskaf, K. A., & Selim, A. S. (2017). Fuzzy mobile expert system for academic advising. In 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE) (pp. 1–5). IEEE. <https://doi.org/10.1109/CCECE.2017.7946846>
- Deriu, J., Rodrigo, A., Otegi, A., Echegoyen, G., Rosset, S., Agirre, E., & Cieliebak, M. (2021). Survey on evaluation methods for dialogue systems. *Artificial Intelligence Review*, 54, 755–810. <https://doi.org/10.1007/s10462-020-09866-x>
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (Vol. 1, pp. 4171–4186). Association for Computational Linguistics. <https://doi.org/10.18653/v1/N19-1423>
- Gao, C., Lei, W., He, X., de Rijke, M., & Chua, T.-S. (2021). Advances and challenges in conversational recommender systems: A survey. *AI Open*, 2, 100–126. <https://doi.org/10.1016/j.aiopen.2021.06.002>
- Hou, Y., Li, J., He, Z., Yan, A., Chen, X., & McAuley, J. (2024). Bridging language and items for retrieval and recommendation. *arXiv*. <https://doi.org/10.48550/arXiv.2403.03952>
- Jannach, D., Manzoor, A., Cai, W., & Chen, L. (2021). A survey on conversational recommender systems. *ACM Computing Surveys*, 54(5), Article 105. <https://doi.org/10.1145/3453154>
- Kulkarni, M., Kim, K., Garera, N., & Trivedi, A. (2023). Label efficient semi-supervised conversational intent classification. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Industry Track)* (pp. 96–102). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2023.acl-industry.11>
- Larson, S., Mahendran, A., Peper, J. J., Clarke, C., Lee, A., Hill, P., Kummerfeld, J. K., Leach, K., Laurenzano, M. A., Tang, L., & Mars, J. (2019). An evaluation dataset for intent classification and out-of-scope prediction. In *Proceedings of the 2019 Conference*

on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (pp. 1311–1316). Association for Computational Linguistics. <https://doi.org/10.18653/v1/D19-1131>

Mohamed, H., Badawy, O., Eskaf, K., & Zaghloul, M. S. (2020). A virtual assistant chatbot for college advising system. In S. Smys, T. Senjyu, & P. Lafata (Eds.), *Second International Conference on Computer Networks and Communication Technologies (Lecture Notes on Data Engineering and Communications Technologies, Vol. 44, pp. 625–633)*. Springer, Cham. https://doi.org/10.1007/978-3-030-37051-0_70

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5), 513–523. [https://doi.org/10.1016/0306-4573\(88\)90021-0](https://doi.org/10.1016/0306-4573(88)90021-0)

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. arXiv. <https://doi.org/10.48550/arXiv.1910.01108>

Appendix A: Intent Labels and Operational Meanings

This appendix lists the twenty intent labels implemented by the classifier and the operational meaning used by the routing layer. The labels define the bounded functional scope of the evaluated chatbot.

Table (A.1): Intent labels and operational meanings

Intent Label	Operational Meaning
account management	Password, profile, and account-support requests.
add to cart	Add a specified product to the prototype cart.
billing and payment	Payment-method and billing questions.
compare products	Compare two selected or referenced products.
customer care support	General store-support assistance.
delivery options	Available or simulated delivery methods and times.
issue support	Support for damaged, incorrect, or problematic products.
order management	Manage or inspect existing orders.
order tracking	Track an order reference.
out of scope	Requests outside shopping, products, orders, delivery, payment, and support.
place order	Checkout or order-placement requests.
price filter	Find products under an explicit or inferred price limit.
product details	Retrieve detailed information for a selected product.
product inquiry	Ask whether a product or product type is available.
product recommendation	Request suitable or highly rated products.
product search	Search for products by product, brand, or title terms.
remove from cart	Remove an item from the prototype cart.
shipping cost	Ask about shipping fees.
store information	Ask about the store or its general policies.
submit feedback	Submit or discuss feedback about the store experience.

Appendix B: Executed Chatbot Scenario Matrix

The following matrix records the selected end-to-end chatbot scenarios used in the thesis. Raw intent and final route are retained separately so that deterministic corrections are not reported as additional classifier accuracy.

Table 21 -B.1 Executed chatbot scenarios

ID	User Request	Raw Intent -> Final Route	Confidence	Observed Output
CT-01	Waterproof mascara under \$20	price_filter	98.6%	6 product cards
CT-02	Niacinamide serum for dry skin	product_search	47.2%	6 product cards
CT-03	Compare two eyeliners	compare_products	98.8%	2 product cards
CT-04	Shipping cost	shipping_cost	99.2%	Policy/support response
CT-05	Track ORD-123456	order_tracking	99.1%	Policy/support response
CT-06	Damaged product support	issue_support	99.1%	Policy/support response
CT-07	Visa payment	billing_and_payment	94.4%	Policy/support response
CT-08	Football joke	out_of_scope	99.5%	Policy/support response
CT-09	Misspelled mascara under \$15	remove_from_cart -> price filter	80.5%	6 product cards
CT-10	Oral-B toothbrushes	product_inquiry	90.7%	6 product cards
CT-11	Checkout request	billing_and_payment -> place_order	97.7%	Policy/support response
CT-12	Password reset	account_management	99.2%	Policy/support response

Appendix C: Final Runtime Validation

The final focused runtime validation contained 15 checks and all 15 passed. The suite verifies service health, the 8,000-record catalog, product categories, availability of both trained models, price-constrained recommendation, comparison uniqueness, typographical-error recovery, shipping behavior, negative-budget handling, vague-request clarification, out-of-catalog and out-of-scope handling, blank-message rejection, unsupported model-key rejection, and overlong-message rejection. It complements, rather than replaces, the 52-case chatbot regression report.

Table 22 - C.1 Final runtime, API, asset, and hidden-issue validation results

#	Verification Check	Result	Evidence
1	Health endpoint and application version	Passed	HTTP 200; status ok; version 1.1.0
2	Catalog contains 8,000 products	Passed	SQLite returned 8,000 product records
3	Catalog categories are available	Passed	Category endpoint returned beauty-product groups
4	DistilBERT and BERT-Tiny load locally	Passed	Both model packages returned valid predictions
5	Price-constrained mascara recommendation	Passed	Returned mascara products within the requested budget
6	Product comparison uses unique visible titles	Passed	Three distinct comparison items were rendered

#	Verification Check	Result	Evidence
7	Typographical-error recovery	Passed	Misspelled request was normalized and routed correctly
8	Shipping-policy request	Passed	Expected policy response returned
9	Negative budget is rejected	Passed	Controlled validation response; no products returned
10	Vague request asks for clarification	Passed	System requested missing product or budget details
11	Out-of-catalog product request	Passed	No unrelated beauty-product fallback was returned
12	Out-of-scope request	Passed	Bounded-domain response returned
13	Blank chat message is rejected	Passed	HTTP 422 validation response
14	Unsupported model key is rejected	Passed	HTTP 422 validation response
15	Overlong chat message is rejected	Passed	HTTP 422 validation response

The detailed results are retained in the project audit reports and the separate CHATBOT_FULL_TEST_REPORT.json and CHATBOT_FULL_TEST_REPORT.md files. The six final interface screenshots embedded in Figures (15)–(20) provide visual evidence for the approved identity, successful recommendation and comparison behavior, typographical-error recovery, and controlled validation states.

Appendix D: Installation and Execution Guide

The following steps reproduce the evaluated local execution. Paths may be adjusted for Windows, Linux, or macOS, but the model folders and SQLite database must remain in the locations expected by the backend configuration.

D.1 Backend

```
python -m venv .venv
# Windows
.venv\Scripts\activate
# Linux/macOS
source .venv/bin/activate
pip install -r requirements.txt
python backend/run_server.py
```

The backend initializes the SQLite schema before importing services. The default API is `http://127.0.0.1:8000`, and `GET /api/health` should return status `ok` and version `1.1.0`. Transformers is optional at runtime because the packaged DistilBERT and BERT-Tiny artifacts can be loaded by the project-local PyTorch and safetensors implementation.

D.2 Frontend

```
cd frontend
npm install
npm run dev
```

For source development, Vite serves the interface at `http://127.0.0.1:5173`. The supplied `frontend/dist` directory is the tested prebuilt interface and can be served with any local static server. The hero and product images are local. `VITE_API_BASE_URL` may be used when the backend runs at a different address.

D.3 Verification

```
python backend/scripts/validate_chatbot_expected_results.py
python tests/runtime_validation.py
```

The first command reproduces the 52-case chatbot regression report. The second reproduces the 30-check runtime validation. Review the two Markdown reports, JSON files, final execution logs, `FINAL_RUNTIME_AUDIT_AR.md`, and the screenshot folders before Moodle submission.