TU WIEN Informatics

The Impact of Using Large Language Models on the Performance of Conversational Recommender Systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Masterstudium Business Informatics

eingereicht von

Michael Schmiedmayer, BSc
Matrikelnummer 01610291

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Mag.a rer.nat. Dr.in techn. Julia Neidhardt
Mitwirkung: Projektass. Dipl.-Ing Ahmadou Wagne, B.A.

Wien, 15. Mai 2026

Michael Schmiedmayer	Julia Neidhardt

TU WIEN Informatics

The Impact of Using Large Language Models on the Performance of Conversational Recommender Systems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Master programme Business Informatics

by

Michael Schmiedmayer, BSc
Registration Number 01610291

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Mag.a rer.nat. Dr.in techn. Julia Neidhardt
Assistance: Projektass. Dipl.-Ing Ahmadou Wagne, B.A.

Vienna, May 15, 2026

Michael Schmiedmayer | Julia Neidhardt

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Michael Schmiedmayer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe. Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Overview of Tools used"(„Übersicht verwendeter Hilfsmittel") habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben

Wien, 15. Mai 2026

Michael Schmiedmayer

Acknowledgements

I would like to express my sincere gratitude to my supervisor, *Ass. Prof.in Mag.a rer.nat. Dr.in techn. Julia Neidhardt*, and my co-supervisor, *Projektass. Dipl.-Ing. Ahmadou Wagne, B.A.*, for their continuous support, valuable feedback, and guidance throughout the development of this thesis.

Special thanks go to *Geizhals* for providing the historical user data, which formed the empirical foundation of this research. Furthermore, I am deeply grateful to the TU Wien DataLab for granting me access to their hosted LLMs and computational resources.

Additionally, I would like to thank the members of the *Data Science* research unit, with a special mention to my colleagues at the *CDL-RecSys*, for the insightful discussions and supportive environment.

Finally, I want to thank my family and friends, especially my parents, from the bottom of my heart for their patience, encouragement, and support throughout my entire academic journey.

Kurzfassung

Diese Arbeit untersucht die Integration von Large Language Models (LLMs) in dialogorientierte Empfehlungssysteme (Conversational Recommender Systems, CRS) und evaluiert die Auswirkungen von Modellgröße und Parameterkonfigurationen der LLMs auf die Retrieval-Performance. Über rein künstliche Vergleichstests hinausgehend, führt diese Forschungsarbeit ein neuartiges End-zu-End-Evaluierungsframework ein, das auf realem Nutzerverhalten basiert. Als Datengrundlage für dieses Framework wurde ein kuratierter Datensatz aus 734 Anfrage-Ziel-Paaren, der durch eine Nutzerstudie validiert wurde, direkt aus historischen Klickpfad- und Filterprotokollen der Preisvergleichsplattform *Geizhals* synthetisiert. Empirische Evaluierungen mithilfe dieses Frameworks liefern drei zentrale Erkenntnisse. Erstens weisen LLMs signifikante Limitierungen auf, wenn sie komplexe, mehrstufige Nutzerverhaltensweisen autonom in einzelne natürlichsprachliche Absichten übersetzen, da sie häufig unter Fehlinterpretationen oder Kontextverlust leiden. Zweitens ist die Parametergröße eines Modells beim aktiven Produkt-Abruf via SQL-Erzeugung ein starker Prädiktor für die systemische Zuverlässigkeit und absolute Trefferquoten. Dennoch zeigt das kleinere, auf Programmiercode spezialisierte Modell (`Qwen-Coder-30B`) aufgrund einer engeren Filterlogik eine hohe Präzision und Platzierungsqualität. Schließlich zeigt eine Raster-Such-Analyse, dass die Feinabstimmung der Parameter (Temperatur und Top-p) nur marginale Auswirkungen auf die Gesamtleistung hat. Dies belegt, dass der Erfolg eines CRS primär von einer robusten Anweisungsstruktur und autonomen Rückfallmechanismen abhängt, anstatt von granularen Anpassungen des Auswahlverfahrens.

Abstract

This thesis investigates the integration of Large Language Models (LLMs) into Conversational Recommender Systems (CRS), evaluating the impact of model scale and hyperparameter configurations on retrieval performance. Moving beyond purely synthetic benchmarks, this research introduces a novel, end-to-end evaluation framework grounded in real-world user behaviour. To fuel this framework, a curated dataset of 734 query-target pairs, validated through a user study, was synthesised directly from historical clickstream and filtering logs from the price comparison platform *Geizhals*.

Empirical evaluations using this framework reveal three primary insights. First, LLMs exhibit significant limitations when autonomously translating complex, multi-step user behaviours into single natural language intents, frequently suffering from hallucinations or context loss. Second, during active product retrieval via SQL generation, a model's parameter scale is a strong predictor of systemic reliability and absolute hit rates; however, the smaller, code-specialised model (`Qwen-Coder-30B`) demonstrates superior precision and ranking quality due to more stringent filtering logic. Finally, a grid search analysis shows that hyperparameter tuning (Temperature and Top-p) exerts only a marginal impact on overall performance, showing that CRS success is driven predominantly by robust prompt architecture and autonomous fallback mechanisms rather than granular sampling adjustments.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

> *„The exponential growth in the availability of information brought to us by technological advances brings not only promise, but for many a sense of information overload and frustrations linked to a lack of confidence in using digital tools“*
>
> — International Federation of Library Associations and Institutions (IFLA), IFLA Statement on Digital Literacy Aug. 2017 [LI17]

The rapid and expansive growth of the Internet has resulted in an unparalleled surge in the amount of information available online, highlighting the crucial necessity for the development and implementation of Recommender Systems (RSs). These sophisticated systems are meticulously engineered to sift through and analyse user preference signals, such as historical records of purchases or lists of movies previously watched. By leveraging this data, RSs can identify patterns and preferences unique to each user. Consequently, they can make informed predictions about potential interests and preferences, enhancing user experiences by personalising content and recommendations [Lü+12] [BR20].

Traditional RSs rely on established algorithms and predefined approaches, but they often face limitations in capturing nuanced user preferences and context. The cold start problem, in particular, poses a significant challenge for these systems [GJ17]. To address this problem, Conversational Recommender Systems (CRSs) are used.

According to Wenqiang et al. [Lei+20], CRSs are interactive platforms where users engage in natural language dialogue with the RS, enabling it to elicit their preferences in a detailed manner. In other words, a CRS can be understood as a chatbot that provides

recommendations tailored to the user's specific needs and requirements through single or multiple natural language interactions.

As also stated by Fan et al. [Fan+23], the emergence of Large Language Models (LLMs) like GPT [Ope23], Llama 2 [Tou+23], or PaLM 2 [Ani+23] presents an opportunity to leverage the advanced Natural Language Processing (NLP) capabilities and the vast global knowledge base of these models for a more sophisticated understanding of user behaviours and preferences.

Derived from the previous statements, it is clear that CRSs that are utilising LLMs are becoming more and more important.

Based on this motivation, this research addresses the overarching problem of how to integrate LLMs in CRSs. Moreover, it is crucial to comprehend how the different sizes of these language models influence the performance of CRSs utilising them. In this context, size refers to the number of parameters a model has. The number of parameters correlates directly to the computational requirements needed to run an LLM. The scalability of LLMs will introduce computational challenges. Determining the optimal size that balances computational efficiency with performance gains becomes a pertinent concern.

The knowledge gap, specifically the lack of a systematic comparison in this area, impedes the development of efficient CRSs. To further clarify, because most systems aim for scalability, the absence of such benchmarks makes it difficult to leverage the advantages of LLMs while mitigating potential computational bottlenecks.

Furthermore, exploring various hyperparameter configurations of LLMs is essential for tailoring CRSs to diverse user contexts and preferences. These hyperparameters determine how an LLM answers the queries of a user. This could be the maximum or minimal length of an answer or the "Temperature" which controls the randomness of the answer or the "Top-p" that controls the possibility of the top tokens to be chosen [Ouy+23]. Investigating the nuanced interplay between different hyperparameters and their impact on recommendation performance is vital for optimising the implementation of LLMs in CRSs.

The absence of a systematic exploration of these factors limits the adaptability of CRSs to varying user preferences and contexts, thereby hindering their overall effectiveness.

1.2 Aim of the Work

The primary aim of this research is to systematically evaluate and comprehend the impact of integrating LLMs into CRSs. A CRS is being used, in which certain tasks are handled by an LLM. This system integrates an LLM as one of its components, and this LLM is interchangeable so multiple models can be tested. Based on this system, the following three research questions will be answered:

RQ1: To what extent can an LLM accurately produce question and answer pairs for user simulation in a CRS-context based on behavioural data?

It is challenging to evaluate CRSs as stated by Jannach [Jan23]. For this reason, this thesis aims to create a dataset containing question and answer pairs created from a baseline dataset. This dataset will then be used to evaluate the created system and allow the comparison between the different LLMs and their different configurations by having a ground truth. The evaluation of the dataset will be done similar to the approach proposed by Thomson and Reiter [TR20]. The methodological design of this data generation and the subsequent human evaluation are detailed in Section 4.1 and Section 4.2, while the empirical findings answering this question are presented in Section 5.1.

RQ2: To what extent does the size of an LLM impact the performance of a CRS?

This question not only aims to investigate computational efficiency but also gains or losses in the performance of the recommendations. Insights into the size requirements that optimise RSs for computational efficiencies and accuracy of the recommendations will be provided by systematically exploring different model sizes. The architectural setup for testing varying LLM scales is outlined in Section 4.3, and the comparative results are discussed in Section 5.2.

RQ3: To what extent do various hyperparameters and configurations of LLMs influence the performance of a CRS?

LLMs offer diverse sets of hyperparameters and configurations, which adds complexity to the integration process. This research explores the interaction between different hyperparameters and their impact on recommendation performance. By comprehensively understanding these factors, this research aims to guide the development and deployment of more effective and adaptive RSs that cater to diverse user contexts and preferences. The experimental grid setup is defined in Section 4.3, and the resulting performance metrics are analysed in Section 5.3.

To address these questions, this thesis provides three contributions that advance the current state of CRSs:

- **Methodological:** The development of a data-generation approach that bridges implicit user behaviour (clickstreams, filter usage) and explicit natural language queries. This resulted in a human-validated dataset for evaluating CRSs based on real-world e-commerce interactions.
- **Technical:** The design and implementation of an end-to-end CRS evaluation framework. This pipeline features autonomous SQL generation, self-correcting error-recovery loops (syntax repair and semantic relaxation), and an interchangeable re-ranking mechanism to systematically benchmark different LLMs and hyperparameter configurations.
- **Empirical:** A comprehensive analysis detailing how varying LLM sizes and hyperparameter configurations (Temperature, Top-p) impact retrieval accuracy, ranking precision, and computational latency within a CRS.

1.3 Methodological Approach

For this master's thesis, Design Science Research [Hev07] will be used as a research framework. This includes the following steps:

- **Design as an Artifact:** The artifact in this research is the integration of different LLMs into a CRS. The design involves the development and implementation of a CRS that incorporates state-of-the-art language models such as Llama 2. The artifact will be implemented by following a conventional software development model as suggested by Iivari [Iiv07], which is the traditional software development model.
- **Problem Relevance:** The research addresses the pressing issue of understanding the impact of LLMs on CRSs, a topic of significant relevance in the evolving landscape of recommendation mechanisms. By systematically evaluating this integration, the research aims to provide practical solutions to challenges faced by traditional recommendation approaches.
- **Design Evaluation:** The effectiveness of the designed artifact will be evaluated through comparisons with the ground truth gathered from the user behaviour dataset. This necessitates transforming the data into question and answer pairs, allowing for a uniform evaluation of the CRS against a consistent ground truth across all evaluated LLMs. Evaluation metrics will include Precision@k, Mean Reciprocal Rank (MRR), and Normalised Discounted Cumulative Gain (NDCG)@k in addition to computational efficiency, like running time. The evaluation metrics will be explained in Chapter 4.3. This is going to be a measured and quantitative form of evaluation as defined by Prat et al. [PCA14].
- **Research Contributions:** The research contributes to the academic field by providing insights into optimising CRSs by considering the size, hyperparameters, and configurations of these models. As well as evaluating the use of LLMs in creating datasets for use in evaluating CRSs, containing a natural language question and a ground truth. The findings aim to advance the understanding of how to harness language models for enhanced recommendation performance.
- **Research Rigor:** The research follows a rigorous methodology, incorporating well-established principles of Design Science Research. This includes a systematic iterative approach to artifact development, thorough evaluation, and validation processes.
- **Design as a Search Process:** The available means that are going to be utilised are the available LLMs, the available tools to make them interact with local knowledge (e.g., LangChain [Cha22]), and available recommender libraries in Python (e.g., Pyserini [Lin+21]). Throughout the design, it is important to address legal and ethical considerations, making sure the product follows relevant laws and protects user privacy.

- **Communication of Research:** The research will be communicated through this comprehensive thesis document and several presentations for the Cristian Doppler Lab, company partner, and the TU Wien.

1.4 Structure of the Work

At first, the goal is to provide an overview of the current state of the art in CRSs, LLMs, and CRSs that utilise LLMs. For this purpose, a literature review will be conducted, which is presented in Chapter 2.

Afterwards, the data will be collected and prepared for use in both the user study and the CRS database. The data exploration and preprocessing steps are described in Chapter 3. The newly created dataset will then be evaluated through a user study to ensure the question–answer pairs are practical and realistic. Participants will be asked to rate how plausible each pair is, given a specific user need, and how likely it would be used in a CRS. This evaluation follows the approach proposed by Thomson and Reiter [TR20] and is described in Chapter 4.2.

Next, the CRS will be set up. The system will be based on the InteRecAgent framework proposed by Huang et al. [Hua+23]. An LLM will serve as the main decision-making component and will be able to choose which tools to use. To enable this, libraries such as LangChain [Cha22] or LlamaIndex [Liu22] will be used, allowing the LLM to act as an agent and access tools automatically. These tools will include a product database, a vector store of previous interactions, and a recommender library such as Pyserini [Lin+21]. The exact set of tools did change during development as new options became available and existing ones were updated. The finalised CRS setup is described in Chapter 4.3.

Finally, a framework will be created to evaluate the system. First, LLMs of different sizes will be evaluated and compared. Based on this comparison, one LLM will be selected for further analysis. This selected LLM will then be evaluated using different hyperparameter configurations. The framework will measure the time required to process the same queries for each configuration, along with the other previously mentioned metrics, by comparing the answers generated by the CRS to the ground truth. The overall evaluation framework is described in Chapter 4.3, and the hyperparameter setup is detailed in Chapter 4.3.

CHAPTER 2

Related Work

Before detailing the specific datasets and the system architecture developed to address these research questions, it is essential to contextualise this work within the existing literature. Therefore, a comprehensive overview of the theoretical background and related work concerning CRSs that utilise LLMs will be provided in this chapter.

First, the fundamental principles and established approaches of classical RSs will be examined in Section 2.1.

Afterwards, the focus will shift to the paradigm of CRSs, detailing their interactive mechanisms and how they work, which can be found in Section 2.2.

Next, in Section 2.3, the underlying concepts of LLMs will be described, alongside an exploration of LLM-based agents.

Following this, the intersection of these domains will be explored in Section 2.4 by providing a review of CRSs that utilise LLMs.

Subsequently, the evaluation metrics employed to assess the performance of RSs will be outlined in Section 2.5, alongside an overview of NLP evaluation methodologies. This will provide the necessary theoretical foundation for assessing the LLM-generated outputs.

2.1 Recommender Systems

RSs are defined as software tools and computational techniques designed to provide users with personalised item suggestions that are highly likely to be of interest or utility. Within this context, an “item” can represent a diverse array of entities, ranging from multimedia content, such as movies and musical tracks, to physical consumer goods, such as vehicles and computer hardware. These systems primarily try to facilitate and streamline the user’s decision-making process. This support is particularly important in scenarios where

the user lacks domain expertise or familiarity with the specific characteristics of the available or searched-for items [RRS10].

As mentioned by Bobadilla et al. [Bob+13], the generation of recommendations by an RS is based upon a combination of the following considerations:

- **The available data:** The specific type of data present in the database, which may include user ratings, demographic registration information, rankable item features and content, social relationships among users, and/or location-aware information.
- **The filtering algorithm:** The underlying algorithm used to filter the data, such as demographic, content-based, collaborative, or hybrid filtering approaches.
- **The chosen model:** The architectural model selected, which can be categorised as either "memory-based" (relying on the direct use of data) or "model-based" (generated by abstracting patterns from the data).
- **The employed techniques:** The computational methodologies applied, encompassing probabilistic approaches, Bayesian networks, nearest neighbour algorithms, bio-inspired algorithms (e.g., neural networks and genetic algorithms), fuzzy models, and dimensionality reduction techniques like Singular Value Decomposition to mitigate data sparsity.
- **Database characteristics:** The inherent sparsity level of the database and the desired scalability of the overall system.
- **System performance:** The operational efficiency of the system, specifically concerning time complexity and memory consumption.
- **The primary objective:** The specific goal sought by the system, such as calculating exact rating predictions or generating top-N recommendation lists.
- **The quality of results:** The desired characteristics of the final recommendations, typically evaluated through metrics such as novelty, coverage, and precision.

Among these considerations, the choice of filtering algorithm is arguably the most defining characteristic of any RS. Expanding upon this specific dimension, Burke [Bur+00] categorises the core filtering algorithms into five distinct paradigms:

- **Collaborative Filtering**: This approach evaluates items based on the collective opinions of user communities, effectively scaling traditional word-of-mouth recommendations through computational means. By constructing a user-item ratings matrix, the system identifies behavioural patterns to recommend items favoured by individuals with similar tastes. The underlying rating data can be gathered explicitly (e.g., through direct user evaluations) or implicitly (e.g., inferred from browsing or purchasing actions), and may be structured in scalar, binary, or unary formats [Sch+07].

- **Content-based Filtering**: This method focuses on the intrinsic attributes of the items themselves, generating recommendations by matching the features of unseen items with the features of items the user has previously evaluated positively [PB07].
- **Demographic RSs**: These systems utilise explicit demographic profiles, such as age, gender, occupation, or geographic location, to categorise users and identify overarching preference trends within specific demographic segments [Al-16].
- **Knowledge-based RSs**: Rather than relying on a repository of historical user ratings, thereby avoiding the prevalent "ramp-up" or cold-start problem associated with other filtering methods, this approach leverages explicit domain knowledge. It reasons about how specific item features satisfy particular user constraints, frequently employing techniques such as case-based reasoning. These systems typically elicit preferences dynamically, allowing users to iteratively refine their search criteria through interactive dialogues [Bur+00].
- **Hybrid Systems**: A hybrid approach strategically combines two or more of the aforementioned techniques to mitigate the inherent limitations of individual models (such as data sparsity or the cold-start problem) and to systematically enhance overall recommendation accuracy [Bur07].

2.2 Conversational Recommender Systems

A foundational definition of a CRS is provided by Jannach et al.:

> A CRS is a software system that supports its users in achieving recommendation-related goals through a multi-turn dialogue [Jan+21].

It is important to note that this interactive dialogue can manifest through various modalities, including natural language interfaces, structured forms, or application-specific graphical inputs [Jan+21].

In contrast to this multi-turn paradigm, the majority of traditional RSs operate in a static, "one-shot" scenario. This single-interaction model makes it exceedingly difficult for traditional approaches to reliably estimate user preferences when historical interaction data is scarce, a challenge particularly prevalent within high-involvement domains where user purchases are infrequent. Furthermore, one-shot systems struggle to automatically deduce a user's highly variable, context-dependent needs during a specific browsing session. They also operate on the frequently flawed assumption that users arrive at a platform with fully formed preferences, or they miss the fact that a user's preferences change over time, meaning that historical data is no longer accurate. In reality, users often construct their preferences and learn about the available item space dynamically during the decision-making process itself. By facilitating a continuous, task-oriented dialogue, CRSs directly address these limitations, enabling the system to actively elicit

current preferences, provide transparent explanations for suggestions, and immediately adapt to user feedback [Lei+20; Jan+21].

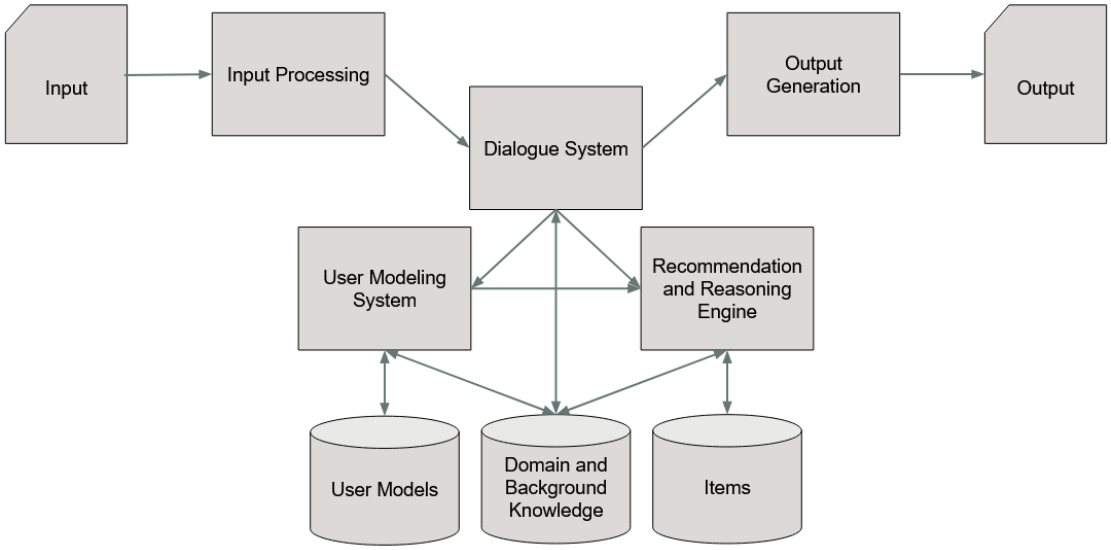


Figure 2.1: Overview of a common CRS. Source: [Jan23]

To provide further context regarding the structural evolution of these systems, Figure 2.1 illustrates the standard architectural blueprint utilised in the development of CRSs. Within this widely adopted paradigm, the underlying architecture can be implemented either as a pipeline of distinctly segmented, specialised functional components, or alternatively, trained as a unified, end-to-end system [Jan23].

The conversational flow begins with user input, which is initially parsed by an *input processing* module before being forwarded to a central *dialogue system*. This *dialogue system* acts as the primary orchestrator of the interaction. To formulate an appropriate response and suggestion, it interfaces simultaneously with two distinct computational components: a *user modeling system*, which is responsible for dynamically tracking and updating user preferences, and a *recommendation and reasoning engine*, which calculates the optimal item suggestions [Jan23].

These operational modules are interconnected with, and reliant upon, structured underlying databases. As depicted, the data storage can be segregated into distinct repositories for *user models*, *domain and background knowledge*, and the *catalog of items*. Once the recommendation logic is resolved, the resulting data is routed through an *output generation* module to be formulated and presented back to the user [Jan23].

For this thesis, we look at the segmented approach as the standard baseline for CRS design. An additional example of a methodology employing a similar, multi-component architecture is detailed in the work by Thompson et al. [TGL04].

Furthermore, evaluating the efficacy of such intricate systems presents significant chal-

lenges, particularly when real-world interaction data is scarce or unavailable. In the absence of live user data, a standard methodological approach for system evaluation is the employment of user simulation techniques, a strategy effectively demonstrated in the research conducted by Sun et al. [SZ18].

2.3 Large Language Models

The foundational architecture underpinning modern LLMs is the Transformer, originally introduced by Vaswani et al. [Vas+23], as illustrated in Figure 2.2. Deviating from previous sequence transduction models that relied heavily on complex recurrent neural networks or convolutional neural networks, the Transformer architecture completely discards recurrence in favour of a mechanism known as attention.

The model is structured around a bipartite encoder-decoder framework:

- **The Encoder:** This component processes an input sequence into a continuous contextual representation. It consists of a stack of identical layers, each containing two primary sub-layers: a multi-head self-attention mechanism and a fully connected position-wise feed-forward network.
- **The Decoder:** The decoder generates the final output sequence auto-regressively, predicting one token at a time. It mirrors the encoder's architecture but integrates a third sub-layer: an encoder-decoder attention mechanism that allows it to focus on relevant parts of the encoder's output. Furthermore, its self-attention layer is "masked" to prevent the model from attending to future, ungenerated tokens during the prediction phase.

At the core of this architecture is the scaled dot-product attention function, which calculates the relevance between different words by mapping a query and a set of key-value pairs to an output. Instead of computing this single attention function once, the Transformer utilises multi-head attention, running multiple attention layers in parallel. This allows the model to simultaneously process information from different representational subspaces [Vas+23].

Finally, because the architecture lacks inherent sequence tracking (due to the absence of recurrence), it employs positional encodings. These mathematically generated signals are injected into the input embeddings, providing the model with essential contextual information regarding the absolute and relative order of the tokens within the sequence [Vas+23].

Building upon this foundational Transformer architecture, several prominent LLMs have been developed, including the generative GPT series [Ope23], Llama 2 [Tou+23], PaLM 2 [Ani+23], and the bidirectional BERT [Dev+19].

Beyond variations in the fundamental encoder-decoder configuration, an additional crucial architectural distinction within modern LLMs pertains to parameter utilisation.

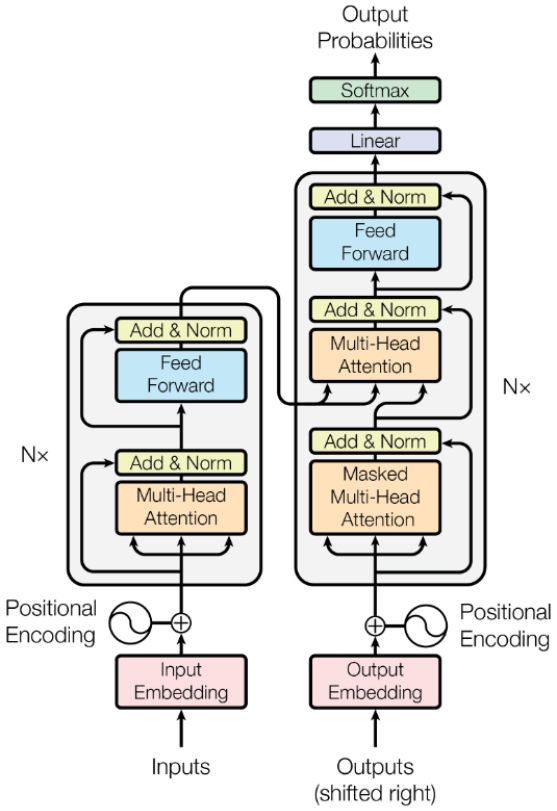


Figure 2.2: Overview of the Transformer model architecture. Source: [Vas+23]

As delineated by Wang et al. [Wan+23a], these models can be broadly categorised into two structural paradigms:

- **Dense Models:** This represents the traditional architectural paradigm where the entire neural network, meaning all model parameters, is activated and utilised for every single computational step and input token. While straightforward and stable to train, scaling up the capacity of dense models inherently demands a proportional, and often prohibitive, increase in computational cost during both training and inference.

- **(Mixture-of-Experts (MoE))**: In contrast to dense networks, MoE models utilise a conditional computation strategy. The architecture consists of multiple specialised sub-networks, termed "experts," managed by a trainable gating or routing mechanism. For any given input token, this routing network dynamically activates only a small, highly relevant subset of the experts. This sparse activation allows the model to dramatically scale up its total parameter count and representational capacity without incurring a linear increase in computational overhead [Sha+17].

Extensive research has consistently demonstrated that these LLMs possess exceptional proficiency in both natural language understanding and generation [Rai+24; Mye+24; Wan+23a; Had+23; AP25].

2.3.1 Large Language Models - Agents

Despite their remarkable capabilities, LLMs exhibit notable limitations when presented with queries that fall outside their training distribution, frequently resulting in the generation of plausible but factually incorrect information, a phenomenon commonly referred to as "hallucination". Furthermore, because their parametric memory is static post-training, seamlessly integrating novel, up-to-date information into their internal knowledge base remains computationally challenging [Rai+24; Had+23; Ji+23].

A promising approach to mitigate these limitations is the deployment of autonomous agents. In contemporary Artificial Intelligence (AI) research, an agent is defined as a goal-oriented computational entity that leverages an LLM as its central cognitive processing unit. Unlike standard conversational models that solely generate text, an agent possesses the capacity to autonomously perceive its environment, formulate multi-step plans through complex reasoning, and execute tangible actions, such as querying databases, retrieving external knowledge, or utilising software tools, to fulfill specific objectives [Wan+24; FTD25; Zha+25].

2.4 Conversational Recommender Systems utilising Large Language Models

The incorporation of LLMs into CRSs represents a recent and innovative paradigm shift that has primarily emerged within the past two years. The novelty of this intersection is evidenced by the absence of LLMs in comprehensive CRS surveys conducted as recently as May 2021, such as the work by Jannach et al. [Jan+21].

Furthermore, research conducted by Di Palma [Di 23] has demonstrated that LLMs can be effectively utilised as stand-alone RSs, achieving competitive performance when compared to more traditional, established methodologies.

Given the inherent complexity of the traditional CRS architecture depicted in Figure 2.1, alongside the advanced natural language understanding capabilities of LLMs detailed in Section 2.3, it becomes evident that integrating an LLM can significantly streamline these multi-component pipelines and enhance overall system efficacy.

In alignment with this perspective, subsequent literature, including research by Fan et al. [Fan+23], asserts that the integration of LLMs has significantly influenced the evolution of CRSs. Models such as GPT-4 [Ope23], BERT [Dev+19], and T5 [Raf+23] have revolutionised the field through their advanced capacity to comprehend and generate human-like text. This textual knowledge has been effectively leveraged to enhance both the performance and the capabilities of modern CRSs.

A prominent example of applying LLMs within the recommendation context is presented by Hou et al. [Hou+23]. In their methodology, LLMs are deployed as zero-shot rankers, meaning the models are not fine-tuned on the specific dataset used for the recommendation task. Distinct from the architecture proposed in this thesis, their approach utilises the LLM as an isolated, stand-alone decision-maker. Nevertheless, their findings in this application emphasise that enhanced recommendation performance is consistently achieved as model size increases.

However, such zero-shot approaches are inherently constrained by the LLM's maximum context window or prompt size. This limitation often leads to potential errors or hallucinations when responding to queries concerning items not explicitly detailed within the prompt. To mitigate these hallucinations and reduce the generation of incorrect information, the integration of a local knowledge base is frequently employed, as proposed by Peng et al. [Pen+23].

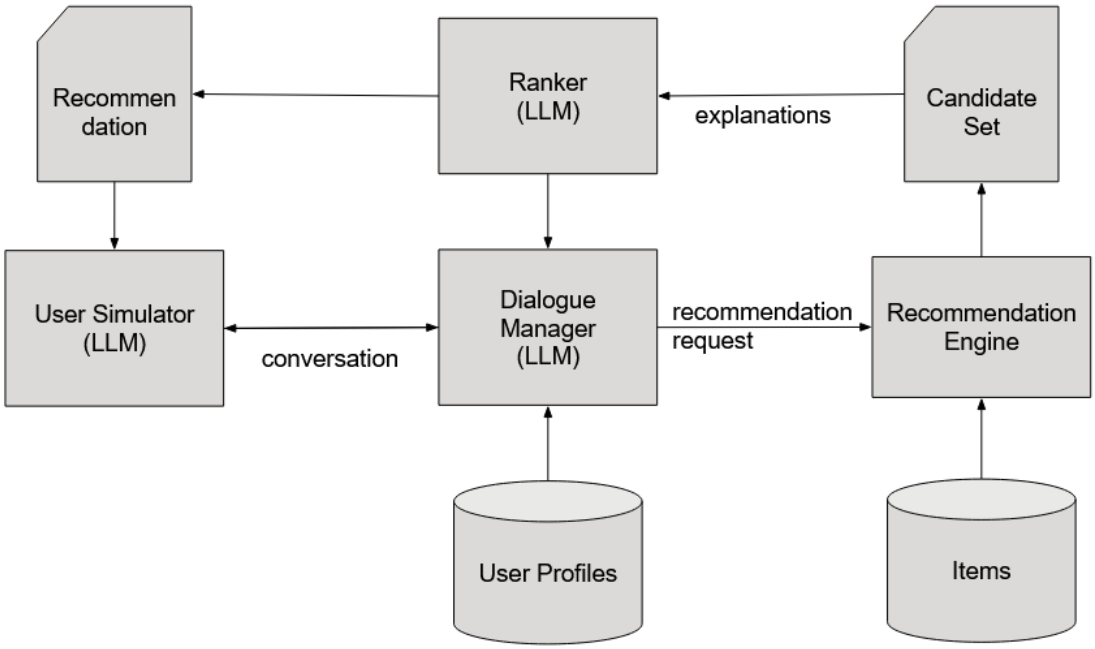


Figure 2.3: Overview of the RecLLM system architecture. Source: [Fri+23]

An implementation of a CRS that successfully combines LLMs with local knowledge is RecLLM, introduced by Friedman et al. [Fri+23] and visually represented in Figure 2.3. This system incorporates multiple distinct modules, including dialogue management, personalised recommendation rankers, and user simulators, each independently powered by LLMs. In contrast, the approach detailed in this thesis simplifies the system architecture by establishing a single LLM as the central decision-making agent. This agent is granted direct access to various external tools, differentiating it from the segmented, multi-module architecture utilised in RecLLM [Fri+23].

The methodology most closely aligned with the system proposed in this thesis is the RecAgent framework developed by Huang et al. [Hua+23], as illustrated in Figure 2.4. In their research, the LLM similarly functions as the core decision-maker, equipped with the ability to dynamically access multiple tools and execute a logical chain of functions to ascertain the optimal item to recommend to the user.

To achieve this, the RecAgent architecture integrates several key components to form a streamlined conversational pipeline. At its core is a *central decision maker*, powered by an LLM, which processes conversational input from the user (or an LLM-based user simulator) to formulate a specific, actionable intention. This intention triggers a *structured execution chain* that guides the LLM's reasoning process through a logical sequence: *initialising the state, leveraging dynamic demonstrations, planning, executing, and reflecting*. During the execution phase, the system dynamically interacts with a specialised *tools* module, comprising *query, retrieval, and ranking functions*. These *tools* iteratively process and refine potential items stored within an underlying *candidate memory bus*. If the system's reflection determines that the current candidate items are insufficient to meet the user's needs, it loops back to adjust its *reasoning* and *tools* execution. Once a satisfactory result is achieved, a final recommendation is generated and returned to the *central decision maker* to present to the user. This architecture effectively streamlines the complex, multi-modular pipelines seen in previous works by utilising a central LLM agent that directly orchestrates tool access and sequential reasoning [Hua+23].

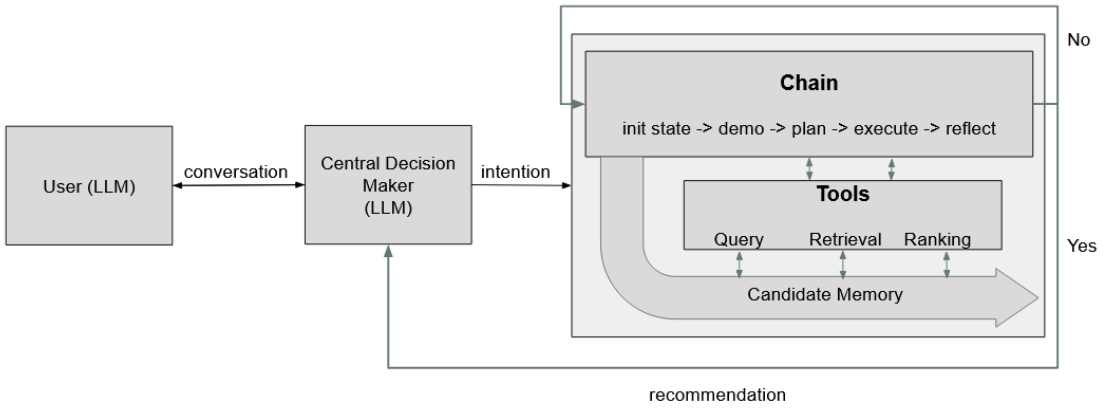


Figure 2.4: Overview of the RecAgent framework. Source: [Hua+23]

Furthermore, both Friedman et al. [Fri+23] and Huang et al. [Hua+23] employ LLMs for the purpose of user simulation within their respective systems. In both frameworks, the models are instructed to act as simulated users through the use of carefully crafted, persona-based prompts. For instance, the system might instruct the LLM to adopt a highly specific persona, such as a twenty-year-old man who enjoys painting and video games. Building upon this foundational concept, Huang et al. [Hua+23] also incorporate

one-turn recommendations, wherein these prompts are dynamically generated based on the user's historical interaction data.

The methodology of utilising LLMs as user simulators in a CRS context is further formalised by Wang et al. [Wan+23b]. In their proposed evaluation framework, denoted as *iEvaLM* and illustrated in Figure 2.5, an LLM is deployed to generate close-to-real user responses. This is achieved by utilising an *existing CRS dataset* to *init* a *user simulator (LLM)* with specific *persona* and *behaviour rules*, while also using the data to *train* the *CRS*. To construct realistic simulated users, ground-truth target items are embedded into templates to *start from an existing conversation*. The system is specifically designed to evaluate interactions across two settings: *free-form chit-chat* and *attribute-based question answering*. A notable drawback to this approach, however, is the necessity of creating handcrafted prompt templates for use in the evaluations.

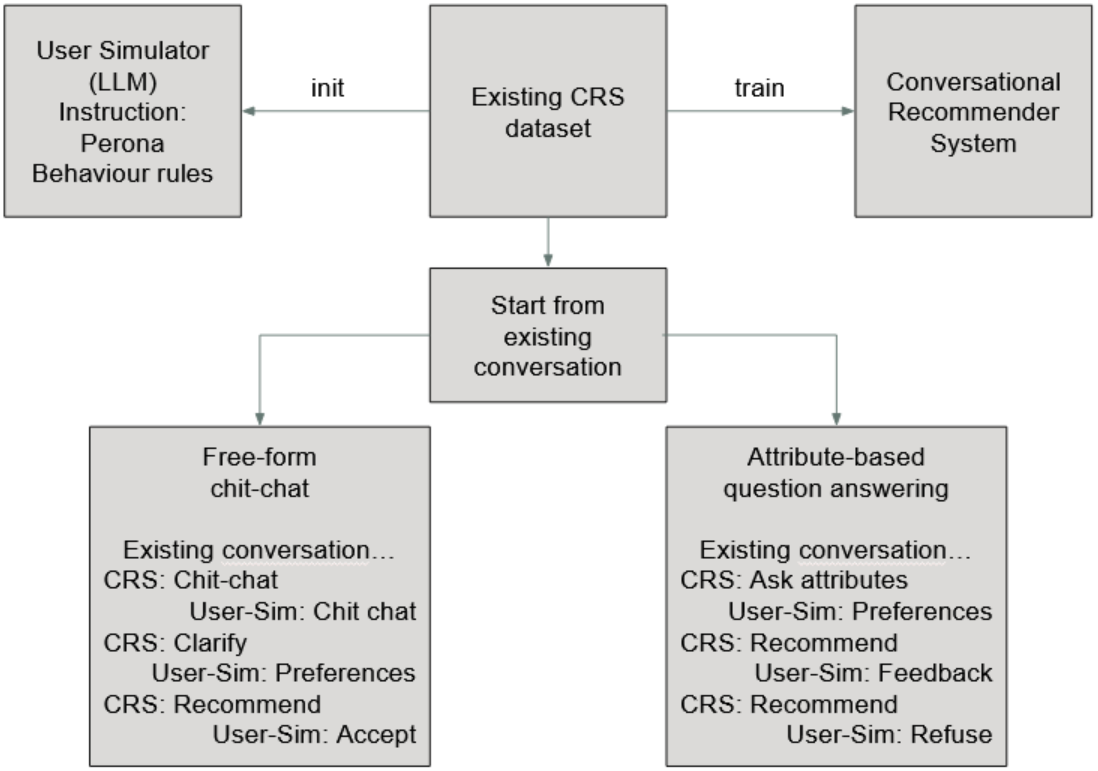


Figure 2.5: Overview of the iEvaLM evaluation approach. Source: [Wan+23b]

While the generated user simulations in these frameworks demonstrate high fidelity, exhaustively verifying their realism remains a critical challenge. To address this, Friedman et al. [Fri+23] outline three potential methodological approaches to systematically assess the realism of the simulated dialogue. These proposed evaluation methods encompass:

1. Conducting a direct comparative analysis between the synthetically generated conversations and actual human-to-human dialogues to identify structural or linguistic discrepancies.

2. Training a distinct discriminator model specifically tasked with classifying and distinguishing between real and simulated interactions.

3. Employing intent classification techniques, such as assigning specific labels to user intents, to systematically compare the distribution and characteristics of the labelled data between the real and simulated datasets.

2.5 Evaluation Metrics

To systematically evaluate the performance of CRSs, several established quantitative metrics will be employed. These metrics are specifically selected to assess the ranking quality and relevance of the generated recommendations. Additionally, to evaluate the output of text generated by an LLM based on data, we identified an approach which is detailed in Section 2.5.4.

2.5.1 Normalised Discounted Cumulative Gain (NDCG)

NDCG is used to evaluate the ranking quality of a recommendation list, specifically accounting for the position of relevant items. As stated by Järvelin and Kekäläinen [JK02], this metric is based on the premise that highly relevant items appearing lower in a search result list should be penalised. The calculation relies on the Discounted Cumulative Gain (DCG) at a particular rank position p, which is defined as follows:

$$\mathrm{DCG}_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

where rel_i represents the graded relevance of the result at position i. To ensure comparability across different queries, the DCG must be normalised using the Ideal Discounted Cumulative Gain (IDCG), which represents the maximum possible DCG (i.e., the items perfectly sorted by ideal relevance):

$$\mathrm{IDCG}_p = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

The final NDCG at rank p is then computed as the ratio of the calculated DCG to the ideal IDCG:

$$\mathrm{NDCG}_p = \frac{\mathrm{DCG}_p}{\mathrm{IDCG}_p}$$

Consequently, the resulting NDCG value lies between 0 and 1, where a score of 1 signifies an ideal ranking.

2.5.2 Precision@k

Precision at k (Precision@k) is a straightforward metric that measures the proportion of recommended items within the top-k set that are genuinely relevant to the user. It provides a direct evaluation of the system's ability to return accurate and useful items within a strictly defined cutoff length k:

$$\text{Precision@}k = \frac{\text{Number of relevant items in the top } k}{k}$$

Precision@k lies between 0 and 1, with 1 being the perfect possible ranking [JP24; Sil+19].

2.5.3 Mean Reciprocal Rank (MRR)

MRR evaluates the system based on the rank position of the first relevant item returned in the recommendation list. The Reciprocal Rank (RR) for a single query is defined as the inverse of the rank position, $rank_i$, of the first relevant item:

$$\text{RR} = \frac{1}{rank_i}$$

The overall MRR is subsequently calculated by averaging the reciprocal ranks across the total number of queries, denoted as $|Q|$:

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

The MRR lies between 0 and 1, with a higher MRR being an indication for a better ranking [JP24].

2.5.4 Natural Language Generation Evaluation

A crucial consideration when deploying these systems is the robust evaluation of the textual output generated by LLMs. To address this, it is necessary to examine established evaluation paradigms within the broader field of NLP. Specifically, this requires a focus on the subfield of Natural Language Generation (NLG), the domain that encompasses and formalises the text-generation capabilities inherent to LLMs [AP25].

At present, human evaluation remains the undisputed gold standard for assessing generated text, primarily because general-purpose automated metrics, such as BLEU [Pap+02], consistently demonstrate weak correlations with actual human judgments. Furthermore,

extensive reviews of the literature highlight that there is currently no single, universally optimal automated evaluation framework [How+20; RB09; GCS23].

For these reasons, we looked at the specific evaluation of the data-to-text aspect within NLG and found the work of Thomson et al. [TR20]. In their study, the authors propose a human evaluation methodology that explicitly focuses on identifying "real-world errors". Rather than penalising generated text solely for containing information absent from the input data, an error is only recorded if the statement is factually incorrect in reality, thereby prioritising actual factual accuracy over strict data adherence. To provide structured, actionable feedback for system developers, they further classify these factual inaccuracies into six pragmatic categories: *incorrect numbers*, *incorrect named entities*, *incorrect words*, *context errors*, *uncheckable statements*, and a general *other* category.

CHAPTER 3

Data Exploration

Having established the theoretical framework and identified the knowledge gaps regarding LLM integration in CRSs, the next step is to examine the specific datasets utilised by the CRS and the subsequent user study. The primary data is sourced from *Geizhals*, a leading Austrian price comparison platform founded in 2000 [Pre26]. Any preprocessing steps, structural modifications to the data, and the rationale behind these changes are detailed in Chapter 4. As an e-commerce aggregator, *Geizhals* provides a rich, multi-dimensional data environment that accurately reflects real-world consumer behaviour and product hierarchies.

The data utilised in this research is categorised into two primary subsets, which serve as the foundation for both system development and evaluation:

- **User Data:** User interaction logs, further detailed in Section 3.1.
- **Product Data:** A product database, further detailed in Section 3.2.

3.1 *Geizhals* User Data

This dataset encompasses anonymised historical interactions, including clickstream data, search queries, and session logs which are based on the database schema from Matomo [Mat26]. Within the scope of this thesis, this data is important in synthesising a realistic dataset for the evaluation of the CRS.

We then categorised the tracking features into logical domains to better understand the underlying user behaviour. The behaviour logs capture granular interactions ranging from initial session entry to specific filtering events and outbound e-commerce leads.

The previously mentioned logical domains are the following four categories: *session & visitor metadata*, *navigation & action metrics*, *event & search tracking*, and *e-commerce*

Table 3.1: Navigation and Action Metrics

Attribute	Description
type	The type of singular action performed: `action` (page view), `ecommerceOrder` (lead), `event` (user click/interaction), `search`, `outlink` (e.g., to a product page on an external site), or `download`.
url / pageTitle	The address and title of the site where the current action was performed.
pageIdAction / idpageview	Unique IDs associated with the specific page action and the broader page view session.
pageviewPosition	The chronological position of the pageview within the current visit.
title / subtitle	Context-dependent labels based on the `type` of action (e.g., page title for `action`, order title for `ecommerceOrder`, or query string for `search`).
dimension1	Custom dimension indicating the page's position within the platform hierarchy (e.g., chosen category). This dimension will be further explained in the product data section, Section 3.2.

Table 3.2: E-commerce and Lead Data (Product Data)

Attribute	Description
productViewName / productViewSku	The displayed name and unique ID of the product currently being viewed.
productViewPrice	The listed price of the viewed product.
productViewCategories	The third-level category of the currently viewed product.
itemDetails	A metadata dictionary containing the lead's `itemSKU`, `itemName`, `itemCategory`, `price`, and `quantity` (defaulted to 1).

& lead data. The following tables outline the key attributes logged within the system. It is important to note that certain fields remained entirely unpopulated throughout the analysed timeframe (September 1, 2023, to September 13, 2023) or are not relevant for our use case. Consequently, to maintain clarity and relevance, these null attributes have been omitted from the following descriptions.

Table 3.3: Event and Search Tracking

Attribute	Description
eventCategory	Classification of the event (e.g., `search`, `Category`, `Compare`, `Productpage.Result`, `Price alert`, `Deals`).
eventAction	The physical action triggering the event (e.g., `click`, `Hover`, `Scroll`, `Submit`, `Filter`).
eventName	Granular metadata providing further context about the tracked event.
dimension3 / **dimension4**	Applied category filters, typically formatted as *thirdLevelCategory_filterID_filterSetting*. These were coded in numbers and subsequently changed back to clear text.
siteSearchKeyword	The literal search query string entered by the user (tracked when `type = 'search'`).

Table 3.4: Session and Visitor Metadata

Attribute	Description
idVisit	Unique identifier for a single visit/session.
country	Origin country of the visit.
timestamp / **serverTimePretty**	The exact time of the interaction, stored both as a Unix timestamp and a formatted string (e.g., Jan 1, 2023 23:59:59).

To gain a foundational understanding of the dataset, we first evaluated the completeness of its various attributes. As illustrated in Figure 3.1, there is significant variance in data density across the dataset. While core tracking identifiers, such as `pageId` and `idpageview`, are nearly 100% complete, many transaction-specific fields, including `revenue`, `orderId`, and `itemDetails`, exhibit extreme sparsity, approaching a 100% missing value rate. Furthermore, interaction-specific fields, such as applied filters, show over 90% missing data. This high degree of sparsity is expected, as many attributes are conditionally recorded only during specific user actions. For instance, the 90% missing data rate in `dimension3` (representing filter usage) simply indicates that filters were actively applied in 10% of the recorded interactions, rather than signifying a failure in data collection. Consequently, the presence or absence of data in these conditionally populated fields serves as a valuable proxy from which average user behaviour can be inferred.

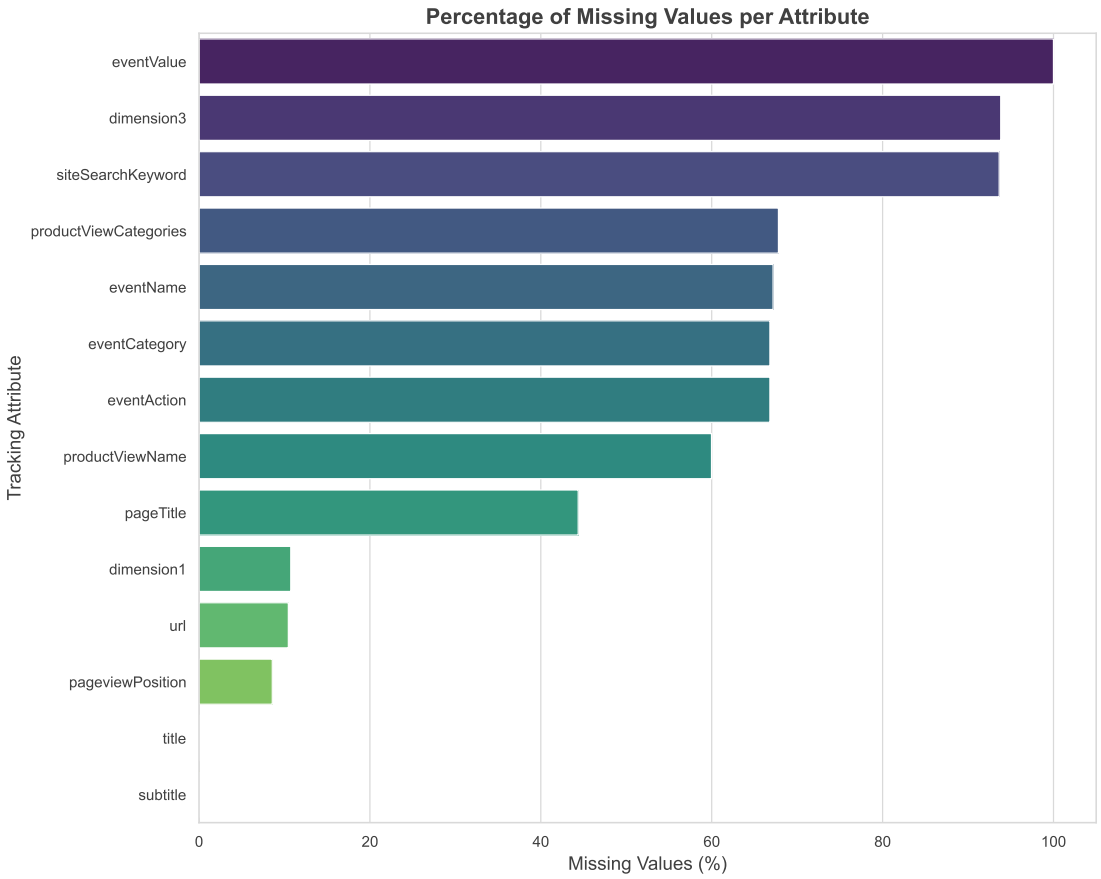


Figure 3.1: Missing values in % for the most important categories.

To further analyse user behaviour during item discovery, we categorised key website interactions into four primary types:

1. Category changes.
2. Filter utilisation.
3. Searches.
4. Viewed product.

The distribution of these interactions provides valuable insights into how consumers navigate the price comparison platform. As shown in Figure 3.2, *product views* (40.02%) and *category changes* (34.58%) constitute the vast majority of logged events.

Conversely, active *searching* and *filter utilisation* occur far less frequently (accounting for only 6.36% and 6.18% of interactions, respectively). This behavioural pattern suggests that users often arrive at the platform with a predetermined goal or land directly within a specific category via external search engines. Consequently, their on-site activity is predominantly focused on browsing and comparing items within those established sections rather than initiating new, exploratory search queries.

Finally, an analysis of outbound clicked product links was conducted to understand conversion pathways. However, due to data limitations and privacy-preserving anonymisation within the dataset, further granular details regarding the specific structure and metadata of these outbound links cannot be disclosed or analysed in depth within the scope of this thesis.

3.2 *Geizhals* Product Data

This subset contains technical specifications, category taxonomies, and pricing information for a wide range of consumer products. This structured data serves as the "knowledge base" for our agent, allowing it to perform precise filtering and retrieval based on specific user requirements (e.g., technical requirements, brand preferences).

The products are organised according to a three-tiered hierarchical taxonomy, structured by increasing levels of granularity:

- **Level 1 (Macro-Category):** Broad, overarching domains encompassing major product spheres, such as Hardware, Telephones, or Audio & HiFi.
- **Level 2 (Sub-Category):** A more granular classification delineating specific product types within a macro-category. Within the *Hardware* domain, for instance, this includes discrete components like Processors, Graphics Cards, or Monitors.
- **Level 3 (Micro-Category):** The most specific taxonomic level, which further refines the product type. For example, within the Processors sub-category, this level differentiates between specific architectures and sockets or use cases, such as HPC-processors.

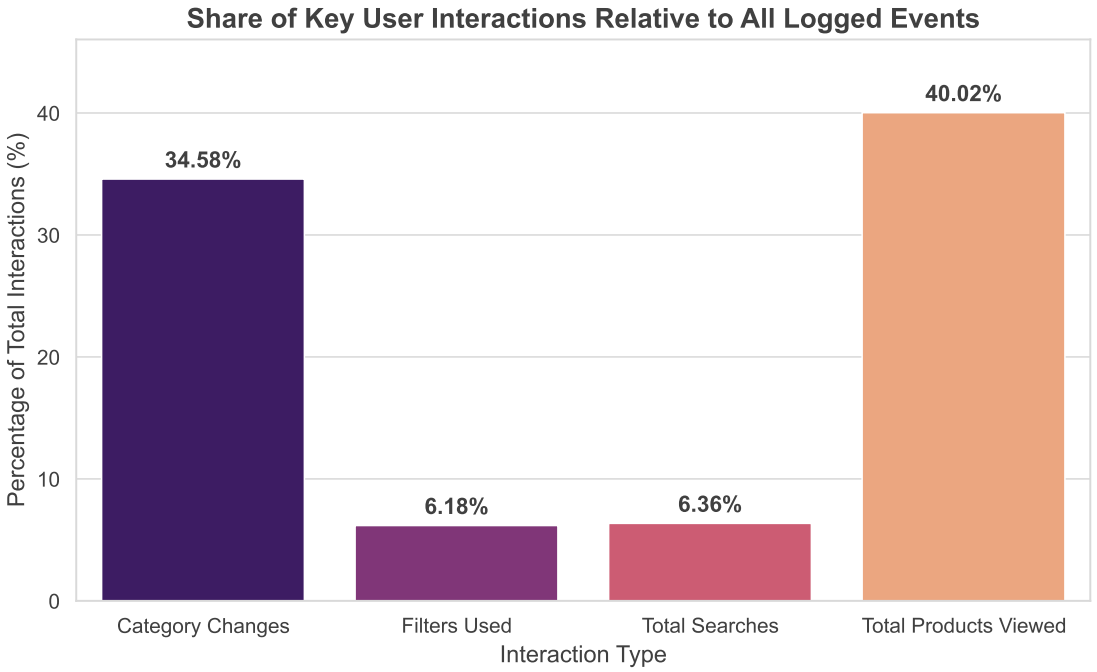


Figure 3.2: Distribution of interaction types.

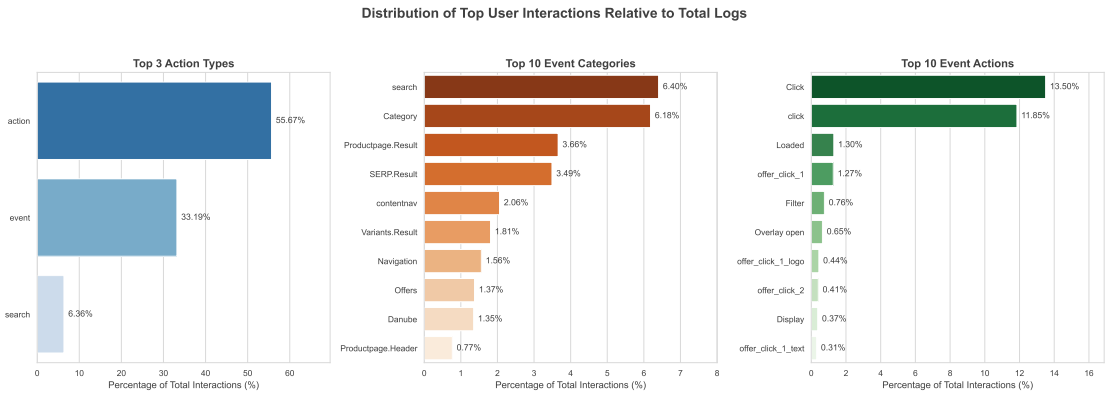


Figure 3.3: Primary interaction distributions showing the top 3 action types, top 10 event categories, and top 10 event actions.

The database comprises several hundred thousand distinct products, each accompanied by its own comprehensive dataset. To optimise computational resources and ensure a focused analysis, this research concentrates on the *Hardware* macro-category, as it represents the most extensive and frequently accessed domain within the platform. The primary attributes utilised from this dataset are detailed in Table 3.5.

Table 3.5: Product Data Table

Attribute	Description
product_link	A direct URL pointing to the specific product page on the *Geizhals* website.
category	The hierarchical taxonomic classification assigning the product to a specific domain. This encompasses all three levels of taxonomic depth (e.g., Hardware / PC Components / RAM).
product	The formal designation of the item, typically encompassing the brand name, model series, and core identifier.
listed_since	A timestamp indicating the exact date the product was initially indexed within the platform's database.
best_price	The lowest aggregate retail price currently available across all monitored external vendors.
product_description	A comprehensive set of technical specifications structured in a JSON format. These features are highly specific to the item's level-three category (e.g., specifying VRAM capacity for a graphics card, or physical dimensions and weight for a laptop).
popularity / rank	A quantitative metric reflecting the product's relative consumer interest, click-through rate, or sales ranking within its respective category.

CHAPTER 4

Methodology

With a clear understanding of the *Geizhals* e-commerce dataset and its inherent behavioural patterns, we can now structure the experimental pipeline designed to answer our overarching research questions. Building upon the data explored in Chapter 3, this chapter details how raw user logs are transformed into conversational queries, how the evaluation of the created dataset is structured, and how the end-to-end CRS architecture is configured for benchmarking. These methodological steps are designed to directly address the central RQs of this thesis: evaluating the LLM's capacity for user intent synthesis (**RQ1**) and assessing the impact of model scale and hyperparameters on CRS retrieval performance (**RQ2** and **RQ3**).

First, Section 4.1 details the **methodological contribution** of this work: a structured data-generation pipeline that transforms implicit user clickstreams into explicit natural language queries.

Subsequently, Section 4.2 focuses on the human-annotated validation of this newly created dataset. This process, which assesses the plausibility and conversational likelihood of the data following the framework proposed by Thomson et al. [TR20], provides the empirical foundation necessary to answer **RQ1**.

Finally, Section 4.3 details the **technical contribution** of this research: the architectural configuration of the CRS. The system is designed as an end-to-end evaluation framework utilising an LLM as the core reasoning engine, augmented with autonomous self-correction mechanisms. The specific implementation details and evaluation metrics, which facilitate the investigation of **RQ2** and **RQ3**, are described in Section 4.3.

4.1 Dataset Generation Setup

For the experimental setup, we utilised the *Geizhals* user data described in Section 3.1. We first provide an overview of our approach, as shown in Figure 4.1, before detailing

the specific steps taken.

We began by converting the log data into plain text to help the LLM better understand the user's actions. Next, we asked the LLM to interpret the user's requirements from this text and translate them into a natural question that could be used in a CRS.

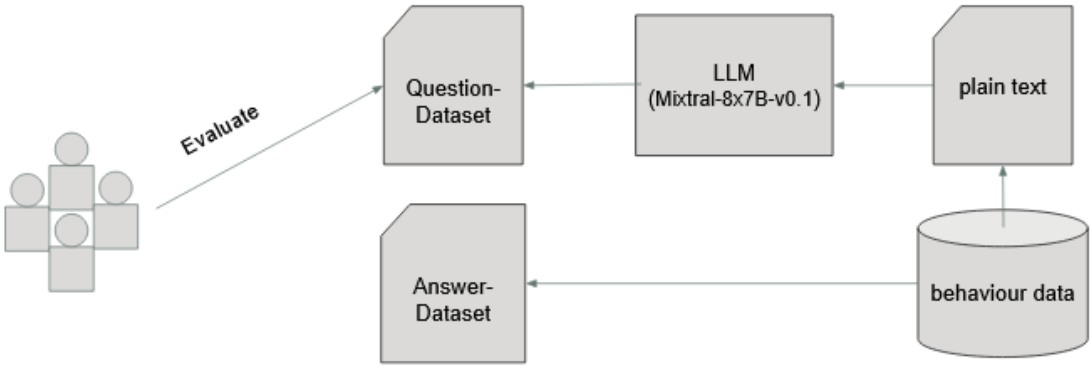


Figure 4.1: Overview of the dataset generation process.

To begin converting the user logs into plain text, we first filtered the dataset to include only interactions from users located in Austria. This step effectively eliminated the majority of automated bots and web crawlers from the dataset while still retaining the vast majority of genuine user interactions.

Next, as mentioned in Section 3.1, we processed the search filters to ensure they were easily readable by the LLM. This involved extracting both the name of the filter and its corresponding selected value, presenting them in a clear, paired format.

To further refine the dataset and focus on highly relevant interactions, we narrowed down the number of sessions. We grouped the interactions by session and filtered specifically for sequences that included a "ProductLinkClicked" or "offer" event within the *Hardware* category. This ensured that the analysed sessions were strictly tied to a user actively exploring hardware products.

Following the initial filtering, the next step involved segmenting the continuous stream of user interactions within each session into discrete, goal-oriented sub-sessions. A sub-session was defined as a sequence of exploratory actions culminating in a terminal event, specifically, when a user either clicked a product link or interacted with an offer.

To achieve this, we generated a new index level that incremented immediately following one of these terminal events. This approach effectively grouped all preceding navigational actions together with the final product interaction they led to. Finally, to ensure data quality and reduce context redundancy for the LLM, we filtered out any identical, consecutive duplicate rows. Each resulting sequence of interactions was subsequently isolated into a dedicated DataFrame for the transformation into plain text.

Once the user sessions were successfully segmented into discrete sub-sessions, the subsequent step involved classifying each session based on the specific exploratory behaviours the user exhibited. These were mentioned in Section 3.1. To achieve this, we analysed the sequence of actions within each grouped DataFrame to detect the presence of the four distinct behavioural dimensions:

1. **Filter utilisation:** Detecting if the user actively applied or changed search filters.
2. **Search queries:** Identifying the use of the platform's manual text search functionality.
3. **Category navigation:** Tracking if the user navigated across different product categories (excluding generic search pages).
4. **Alternative product views:** Monitoring whether the user examined other distinct products before arriving at their final clicked item.

The final step in preparing the user data involved converting the structured tabular logs into natural language narratives. Because the LLM processes text rather than raw data tables, we generated a chronological story of each session, describing exactly what the user did step-by-step (e.g., "The user searched for X" or "The user changed the filter Y to Z").

Furthermore, because the original *Geizhals* platform and its dataset are entirely in German, we translated key attributes, such as category names, filter names, and product titles, into English using the `deep_translator` library [Bac26]. While we maintained both a German and an English version of the narrative, the English translation was essential because the subsequent user study was conducted entirely in English.

The next step involved transforming these segmented interaction DataFrames into natural language questions using the LLM. To optimise computational resources and ensure a high-quality evaluation, we created a specific subset of sessions to be used in the subsequent user study.

To ensure this subset was representative of the diverse ways users interact with the platform, we categorised each session using a behavioural "signature." A signature is a four-bit representation (e.g., `0010`) that acts as a categorical fingerprint, indicating which of the four behavioural dimensions (*filter utilisation*, *search queries*, *category navigation*, and *alternative product views*) were present (`1` for active, `0` for inactive) during the session.

By analysing the frequency of these signatures across the entire dataset, we identified the most common interaction patterns. The resulting distribution, shown in Table 4.1, provided the logical basis for our sampling strategy. This allowed us to select a balanced mix of simple and complex sessions for the user study, ensuring that the LLM's ability to interpret intent was tested against a realistic variety of user behaviours.

Table 4.1: Breakdown of user interaction signatures.

Signature	Active Interactions	Percentage
0010	Category	51.63%
0011	Category, Product	11.12%
1011	Filter, Category, Product	10.62%
0110	Search, Category	8.02%
1010	Filter, Category	7.96%
0111	Search, Category, Product	7.28%
1111	Filter, Search, Category, Product	2.69%
1110	Filter, Search, Category	0.68%
0000	None	0.00%
1000	Filter	0.00%
0100	Search	0.00%
0001	Product	0.00%
1100	Filter, Search	0.00%
1001	Filter, Product	0.00%
0101	Search, Product	0.00%
1101	Filter, Search, Product	0.00%

Based on this foundational distribution, we constructed a stratified sample comprising 150 randomly selected sessions. Rather than sampling strictly proportionally, which would have overwhelmingly skewed the dataset toward simple category navigation (0010), we defined a target distribution to ensure the LLM was evaluated on a diverse array of behavioural patterns. This deliberate oversampling of complex interactions allows for a more robust analysis of the model's reasoning capabilities. The precise sampling quotas utilised for this subset are outlined in Table 4.2.

Table 4.2: Distribution for the 150-session user study subset.

Signature	Active Interactions	Sample Size
0010	Category	35
1010	Filter, Category	20
0110	Search, Category	20
0011	Category, Product	20
1011	Filter, Category, Product	20
0111	Search, Category, Product	20
1111	Filter, Search, Category, Product	10
1110	Filter, Search, Category	5
	Total Sessions	**150**

Furthermore, an analysis of session length, with regard to the number of different

interactions with the platform, revealed a positive correlation with interaction complexity. Specifically, sessions exhibiting multiple behavioural dimensions, such as instances where a user utilised search functions, applied filters, and examined multiple alternative products, were substantially longer than those in which the user solely engaged in basic category navigation.

The final phase of the dataset generation pipeline involved configuring the LLM to synthesise the conversational queries from the processed text. To maintain control over the generation process and ensure data privacy, the LLM was hosted locally. During the selection process for the hosting framework, multiple deployment approaches were evaluated, with KoboldCPP [Los24] and the Oobabooga Text-Generation-WebUI [oob24] emerging as the primary candidates. The Oobabooga WebUI was ultimately selected due to its user-friendly deployment process and its native support for OpenAI-compatible API endpoints, which significantly streamlined programmatic interactions with the model.

Regarding the choice of the foundation model, Mixtral 8x7B-v0.1 [Jia+24] was selected. At the time of implementation, this MoE architecture represented the state-of-the-art among open-weights LLMs, offering an optimal balance between advanced reasoning capabilities and the ability to operate within the memory constraints of the locally available hardware.

The final methodological step involved engineering the system prompt to optimise the text-generation performance of the selected LLM. To achieve this, we employed an iterative prompt refinement strategy. Initial prompt drafts were tested on a representative sample of user sessions, and the model's generated queries were manually evaluated. By systematically identifying recurring errors and deviations from the desired output format, we were able to continuously adjust the instructions to mitigate these issues.

Through an iterative prompt engineering strategy, we refined the model's instructions to mitigate formatting errors and optimise tone (the finalised prompt is available in Appendix 6.2). For each generated natural language question, the specific product clicked at the conclusion of the corresponding historical session was recorded, serving as the definitive ground truth for all subsequent retrieval evaluations required to answer **RQ2** and **RQ3**.

4.2 User Study Setup

With the raw interaction logs transformed into natural language queries, verifying their quality and realism is essential before deployment as a benchmark. This phase details the user study design, which validates the generated dataset and explicitly addresses **RQ1**. We developed a structured evaluation framework, utilising standardised assessment criteria, a predefined error taxonomy, and integrated quality controls, to measure how effectively the model captures user intent. To streamline this manual assessment, we employed *Label Studio* [Hum24] to provide evaluators with a consistent and standardised interface.

The study was organised using the stratified behavioural groups previously established in Section 4.1. To ensure the reliability of the results, we established a rigorous data-cleaning and consensus methodology. First, we integrated five "golden samples" into the evaluation set. These samples acted as attention checks with known, unambiguous ground-truth intents. Participants who failed to correctly identify these samples, or who consistently provided inconsistent responses, were excluded from the final analysis to maintain data integrity.

Furthermore, to establish a robust ground truth for sessions evaluated by multiple participants, a majority voting consensus mechanism was applied. In instances where a clear majority was present, the majority label was adopted. In the event of a tie, where no single majority emerged among an even number of raters, a conservative tie-breaking rule was implemented: the more critical evaluation (i.e., "No") was automatically selected. This approach was chosen to minimise the risk of false-positive assessments, ensuring that only the most structurally sound and plausible queries were included in the subsequent CRS performance benchmarks. To quantify the extent of agreement beyond that expected by chance, inter-rater reliability was measured using Krippendorff's alpha (α) [KRI07].

Having established the composition and consensus rules, we configured the evaluation interface within *Label Studio*. The design of this user study was methodologically informed by the human-evaluation frameworks proposed by Thomson et al. [TR20]. The primary evaluation was structured around two core assessments, as depicted in the evaluator interface in Figure 4.2:

- **Question Representation and Accuracy:** Evaluators were asked to determine if the generated question accurately reflected the user intent demonstrated in the input logs. This involved checking for missing information, incorrect representations, and the presence of extraneous, irrelevant details.
- **Contextual Plausibility:** Evaluators were also asked to assess whether the generated query was realistic, specifically, whether a user exhibiting the displayed behaviour would plausibly ask this question to a CRS.

To gain granular insights into the model's failure modes, we employed conditional logic within the interface. If an evaluator deemed the generated question inaccurate (by selecting "No" to the representation question), a subsequent set of options appeared, requiring them to classify the specific type of error:

- **Incorrect Number:** Numerical values in the generated question that contradict the input data.
- **Incorrect Entity:** Incorrect entities (e.g., product names, brands, categories) mentioned in the question.

- **Missing Number:** Essential numerical constraints from the input data that were omitted by the LLM.
- **Missing Entity:** Important entities or filter attributes from the input data that were omitted.
- **Hallucination:** Fabricated information or constraints present in the question that were entirely absent from the input data.
- **Not Checkable:** Aspects of the question that cannot be readily verified against the provided input data, or would require excessive time to evaluate.
- **Other:** Any other inaccuracies not covered by the preceding categories.

The explicit inclusion of the *hallucination* category was a deliberate design choice aimed at tracking instances where the LLM fabricated context, thereby deepening our understanding of its specific generative weaknesses.

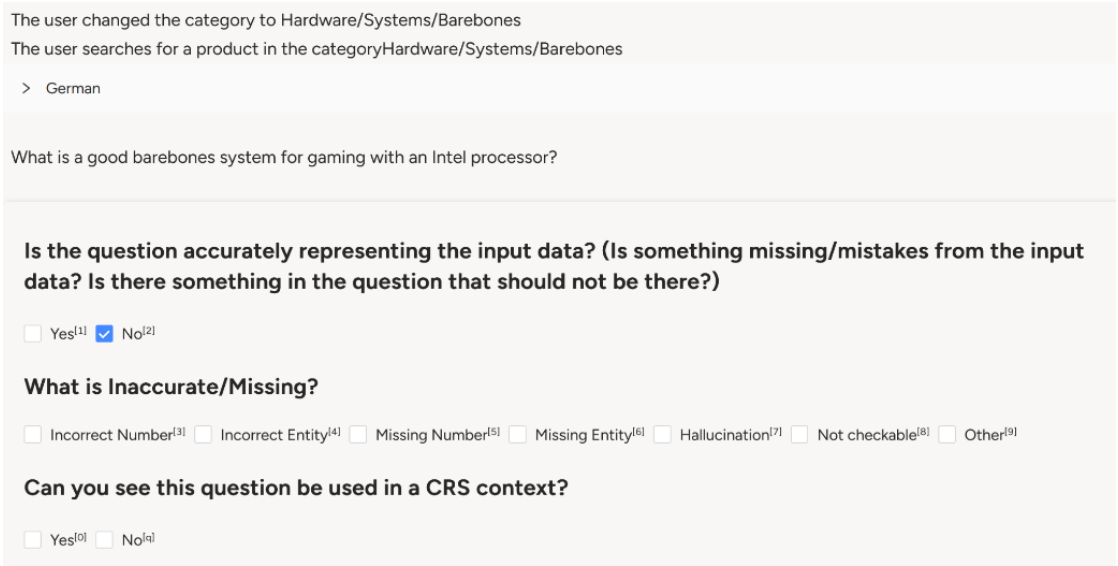


Figure 4.2: The Label Studio interface used for the human evaluation of the LLM-generated questions.

Additionally, to help improve consistency and accuracy across the evaluation process, participants were provided with detailed examples illustrating how to correctly annotate various scenarios. The complete user study guidelines and interface details can be found in Appendix 6.2.

Ultimately, this user study provides the empirical foundation necessary to address **RQ1**, determining whether an LLM can accurately and realistically synthesise conversational queries from raw behavioural data within a CRS context.

4.3 Conversational Recommender System Setup

Once the human evaluation yields a validated dataset of plausible conversational queries with established ground truths, a technical environment is required to process them. Consequently, this final methodological phase establishes the end-to-end architecture of the CRS, providing the necessary technical testbed to investigate the impact of model scaling (**RQ2**) and hyperparameter tuning (**RQ3**). The architectural overview is illustrated in Figure 4.3. The pipeline utilises the validated question dataset as its input. At its core, an LLM functions as the *central decision-maker*, equipped with direct access to a structured *product database* and an *interchangeable re-ranking algorithm*. Ultimately, the system's output recommendations are systematically evaluated against the historical ground-truth targets.

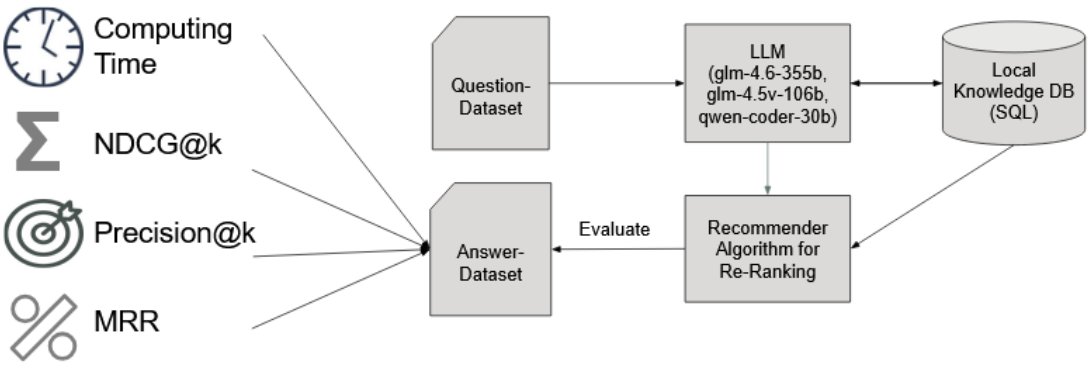


Figure 4.3: Overview of the CRS architecture.

4.3.1 Out-of-the-box Approaches

The initial step involved establishing the underlying database infrastructure with which the LLM would interact. For this purpose, we opted to deploy a self-hosted instance of MariaDB [Mar25], a free and open-source relational database management system (RDBMS). This specific system was selected due to its overall ease of use and its high syntactical compatibility with MySQL, which significantly streamlined both the initial integration and the subsequent query formulation processes. Once deployed, the database was populated with the hardware product catalog previously detailed in Section 3.2.

Following the database configuration, the subsequent step involved establishing a mechanism for the LLM to interface with the stored data. To achieve this, we first evaluated multiple implementation strategies across two prominent orchestration frameworks:

- **LlamaIndex** [Liu22]
  - Agent-based execution
- **LangChain** [Cha22]

- Chain-based execution
- Agent-based execution

To evaluate LangChain's chain-based execution, we tested its ability to autonomously navigate the database schema and retrieve appropriate products. The chain executes a predetermined sequence of thoughts, actions, and observations. For instance, when tasked with finding a 3D printer under €300, the step-by-step execution log, detailed in Appendix 6.2, demonstrates this process. The log illustrates how the agent first lists the available tables, identifies the relevant `3D_Drucker_dr3d` table based on the price constraint, executes a SQL query, and observes the returned product data to formulate a final recommendation. This was relatively fast in the execution timeframe, however, it was very prone to making mistakes and choosing completely wrong tables and/or columns to query for.

Conversely, when testing the LlamaIndex framework, the system successfully formulated the necessary SQL statements to answer complex constraints, such as finding a specific generation of processors within a set price range. However, an analysis of the execution trace, provided in Appendix 6.2, revealed significant performance bottlenecks. The trace shows that while the model successfully generated a complex SQL query filtering by price, category, and architecture, the primary LLM call took over 370 seconds to process the request.

Finally, we evaluated the LangChain agent-based approach. To facilitate autonomous data retrieval, we equipped the agent with distinct operational tools, specifically granting it access to both the relational database and a semantic vector store (the precise function of this vector store is detailed in the subsequent paragraph). However, as illustrated by the execution trace provided in Appendix 6.2, this implementation proved highly unstable. The agent frequently entered repetitive loops, such as continuously listing database tables without progressing, and constructed highly convoluted, non-functional SQL statements laden with redundant `LIKE` clauses, ultimately failing to retrieve the requested hardware recommendations.

To manage the complexity of the underlying MariaDB schema, which encompasses over 200 distinct product categories, and to further improve the performance of the approaches, both the LlamaIndex and LangChain implementations were augmented with a vector database, specifically ChromaDB [Chr25]. A vector database is a specialised storage system designed to efficiently index and retrieve high-dimensional vector embeddings. These embeddings act as dense mathematical representations of text, capturing underlying semantic meaning rather than relying on exact, literal keyword matches.

In the context of this architectural setup, ChromaDB functioned as a semantic router. To populate this vector store, we utilised an LLM to automatically generate detailed textual summaries for each table, outlining their specific contents, schema attributes, and corresponding hardware categories. These LLM-generated descriptions were then embedded and stored within ChromaDB. When a user submitted a natural language

query, the input was first embedded and compared against these stored table descriptions using a cosine similarity search. This intermediate retrieval step allowed the system to identify the most relevant tables and dynamically inject their specific schemas into the primary LLM's context window. By providing this targeted schema information, the vector store significantly narrowed the search space, guiding the central decision-making LLM to construct accurate SQL queries without exceeding token limits or hallucinating non-existent database structures.

These approaches demonstrated a general capability to generate queries and retrieve relevant products. However, we ultimately concluded that these out-of-the-box implementations were unsuitable for our system architecture. The primary disqualifying factor was severe operational inefficiency and extreme latency. As demonstrated in the LlamaIndex trace (Appendix 6.2), the LLM processing time for a single query could exceed 370 seconds. Furthermore, the autonomous agent loops in LangChain consumed excessive token counts and frequently struggled to efficiently navigate the extensive number of tables present in our MariaDB instance. To ensure that these bottlenecks were inherent to the orchestration frameworks and not merely the result of an underperforming foundation model, these tests were replicated across several different LLMs. Across all tested models, the performance issues persisted; consequently, these agentic approaches were deemed fundamentally incompatible with the real-time responsiveness required for a viable CRS in the required context. These models included Llama 2 70B/13B [Tou+23] and Mixtral 8x7B-v0.1 [Jia+24].

4.3.2 CRS Architecture

In light of these limitations, a custom implementation was developed. While conceptually grounded in standard retrieval architectures, this pipeline decomposes the recommendation task into a sequence of tightly controlled, optimised generation and validation phases supported by deterministic helper functions.

LLM Retrieval Pipeline

The complete walk-through of a single query lifecycle (illustrated in Figure 4.4) proceeds through the following phases:

Phase 1: Table Selection and Sanitisation

- **Process Flow:** The pipeline begins by evaluating the user's natural language query directly against all candidate tables to isolate the most relevant product categories. (Note: Vector database routing was tested but omitted, as a highly descriptive table-naming convention proved equally effective and more streamlined).
- **Prompt Integration:** The specific instructions governing this initial routing phase are detailed in Appendix 6.2.

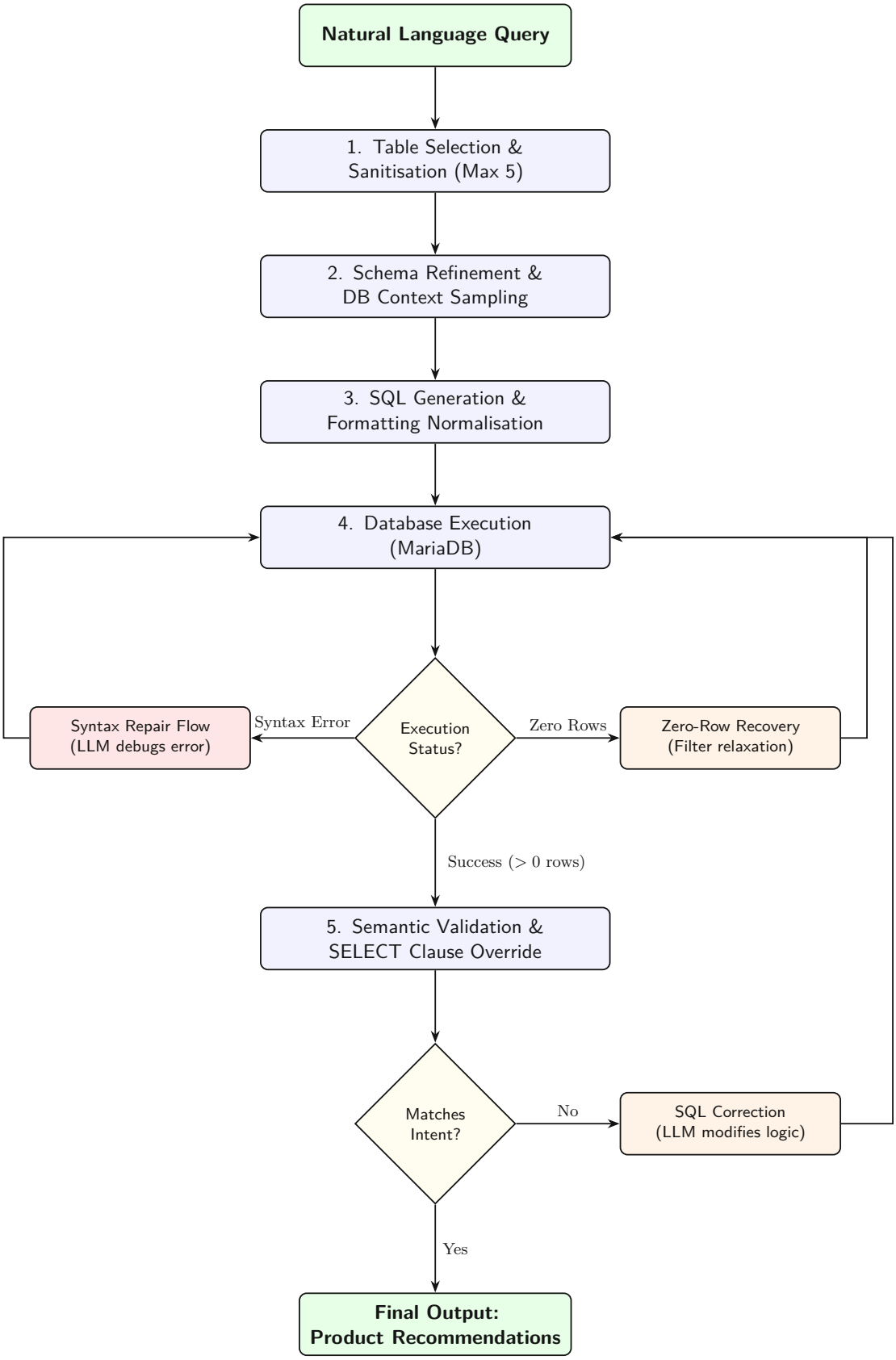


Figure 4.4: Architectural flowchart of the CRS pipeline.

- **Failsafe Mechanism:** To ensure system stability, identified tables undergo programmatic sanitisation. The system verifies their existence within the live MariaDB instance, removes duplicates, and strictly enforces a hard limit of a maximum of five tables. This serves as a critical guardrail against exceeding the LLM's context window.

Phase 2: Schema Refinement and Context Optimisation

- **Process Flow:** Injecting complete schemas can degrade LLM reasoning. Therefore, an auxiliary LLM process evaluates the query against the available column names of the sanitised tables, dynamically selecting only the necessary subset of columns.
- **Prompt Integration:** The prompt driving this targeted column selection is located in Appendix 6.2.
- **Failsafe Mechanism:** The system actively queries the database to extract a maximum of three distinct sample values for the selected columns. This strict limitation on sample retrieval grounds the model in real data formatting while preventing extensive text fields (e.g., product descriptions) from flooding the context window.

Phase 3: SQL Query Generation and Output Normalisation

- **Process Flow:** The optimised schema and data samples are injected into a generation prompt. The LLM maps the user's intent to the provided schema and formulates a MariaDB SQL query.
- **Prompt Integration:** The foundational SQL generation prompt is detailed in Appendix 6.2.
- **Failsafe Mechanism:** Because LLMs frequently append conversational filler or proprietary special tokens (e.g., tool-call markers), a deterministic parser is applied. It systematically strips away Markdown fences, bounds, and whitespace to ensure only a raw, executable SQL statement remains.

Phase 4: Resilient Execution and Strict-Retry Mechanisms
All LLM invocations and database executions are encapsulated within timeout and exponential back-off protocols. Crucially, if execution fails, the system triggers an autonomous self-correction loop targeting two distinct failure modes:

- **Syntax and Execution Error Repair:** If a database error occurs (e.g., hallucinated columns), the system intercepts the trace-back. A dedicated repair prompt (Appendix 6.2) provides the LLM with the failing SQL and the explicit error message for debugging. The corrected output undergoes a regex-based sanity check on the `SELECT` and `FROM` clauses.

- **Semantic Relaxation (Zero-Row Recovery):** If a query executes but returns empty results, the system programmatically dissects the `WHERE` clause, tests individual filters against the database, and feeds the resulting row counts back to the LLM. Recovery proceeds in two tiers:
  1. *Relaxed Retry* (Appendix 6.2): Instructs the model to intelligently adjust the pinpointed problematic conditions.
  2. *Strict Retry* (Appendix 6.2): If the relaxed approach fails, this aggressive fallback forces the model to entirely strip away the restrictive filters causing the zero-row return, guaranteeing a baseline data retrieval.

Phase 5: Semantic Validation and Failsafe Override

- **Process Flow:** Executable SQL may still miss semantic nuances. An LLM evaluator performs a logical cross-check to verify that all constraints mentioned by the user are accurately represented in the query. If discrepancies exist, the LLM outputs a corrected string.
- **Prompt Integration:** The validation instructions are provided in Appendix 6.2.
- **Failsafe Mechanism:** The system dynamically intercepts and overrides the final `SELECT` clause, forcing it to retrieve exactly the required presentation fields (`product` name and `product_link`).

This finalised architectural implementation was the result of a rigorous, iterative refinement process. During development, the system was repeatedly evaluated against a randomised subset of the user query dataset. This empirical testing strategy allowed us to systematically identify, classify, and mitigate the most common execution failures and hallucination patterns. Furthermore, this same iterative optimisation methodology was applied to the prompt engineering process, ensuring that the specific instructions provided to the LLM at each phase of the pipeline were tuned to our database schema and operational requirements.

Used LLMs: During the development phase, the primary LLM deployment infrastructure was migrated to Aqueduct, an AI gateway hosted by TU Wien [TU-26]. This transition enabled the system to leverage significantly greater computational resources and utilise three distinct, high-performance LLMs, each running on different GPUs:

- **GLM-4.6-355B:** A large-scale model deployed across a cluster of four NVIDIA H200 GPUs [TZ25b].
- **GLM-4.5V-106B:** A multimodal variant deployed across four NVIDIA A40 GPUs [TZ25a].
- **Qwen-Coder-30B:** A specialised code-generation model operating on a single NVIDIA A40 GPU [Ali25].

Re-Ranking Approach:

Following the initial retrieval of semantically relevant products via the generated SQL, the subsequent phase required implementing a robust re-ranking mechanism to determine the optimal presentation order for the user. For this, we adopted a strategy similar to the algorithms employed by *Geizhals*. The algorithmic re-ranking is applied to the output, prioritising products based on historical user interaction data (interaction counts) and temporal relevance (listing dates) to ensure that the most highly sought-after and current hardware is presented at the top of the recommendation list.

Using regular expression parsers, the system intercepts the `SELECT` clause of the validated query and dynamically injects specific metadata columns, namely `rank`, `listed_since`, and `count` (a metric representing product popularity through interaction frequency). To ensure the integrity of the query structure, the system accommodates both aliased and un-aliased column formats, and safely injects `NULL` fallbacks if the original schema lacked these specific metadata fields.

The modified statement is then wrapped within a superordinate SQL subquery designed to enforce two strict deterministic constraints at the application level. First, to guarantee temporal consistency across the recommendation dataset, the wrapper applies a temporal filter by parsing the `listed_since` string into a standard datetime object and enforcing a hard cutoff (e.g., excluding products listed after September 2023). This constraint ensures that the retrieved pool accurately reflects the inventory available during the original data-logging timeframe, preventing the inclusion of newer products that the historical users could not have possibly interacted with. Second, the wrapper executes a popularity ranking by appending an `ORDER BY count DESC` clause. This guarantees that the most frequently interacted-with products are surfaced as the primary results.

By abstracting this logic away from the LLM and handling it via deterministic string manipulation, the pipeline guarantees that the final product recommendations are not only semantically accurate but also based on real user behaviour on the *Geizhals* platform.

Evaluation

The final step of the pipeline involves an evaluation to assess the system's overall performance. To measure accuracy, the algorithm compares the final, re-ranked list of product recommendations against the historical ground truth. In the context of this research, the ground truth is strictly defined by a "lead signal", specifically, a documented instance within the dataset where the user demonstrated purchase intent by choosing to view the product on an external seller's website.

This validation phase required a dedicated normalisation layer, as establishing a direct mapping between the products recorded in the historical behavioural logs and those in the live MariaDB schema was non-trivial. Due to inherent structural discrepancies and the highly similar naming conventions of PC hardware, standard fuzzy matching approaches

proved unreliable. Consequently, the pipeline employs a deterministic matching strategy based on sanitised product URLs.

The evaluation script then iterates through the sanitised recommendation list to locate the ground-truth product. If a match is found, its exact presentation index (1-based position) is recorded. To provide a comprehensive and structured assessment of the system's capabilities, these individual ranks and operational logs are aggregated. Table 4.3 provides a compact overview of the purpose and interpretation of all employed evaluation metrics.

Table 4.3: Overview of evaluation metrics and corresponding analysis.

Metric	Purpose & Interpretation	Sections
Hit Rate	Measures absolute retrieval success (i.e., whether the ground truth is present in the returned product list), where higher values indicate a stronger baseline capability of the LLM to locate the correct item.	Sec. 5.2.1, 5.3.2
MRR	Evaluates systemic ranking performance, with higher scores indicating the system efficiently surfaces target products near the top position.	Sec. 2.5.3 (Def.) 5.2.2, 5.3.2
Precision@k	Measures the exactness of the recommendation window, where high values signify successful retrieval within a specific user attention span.	Sec. 2.5.2 (Def.) 5.2.2, 5.3.2
NDCG@k	Evaluates ranking quality by applying a positional penalty, meaning a higher score reflects optimal placement of the ground-truth item within the top-k.	Sec. 2.5.1 (Def.) 5.2.2, 5.3.2
Latency	Assesses real-time operational viability through the running time of the entire pipeline, helping identify trade-offs between model scale and a responsive user experience.	Sec. 5.2.1, 5.3.1
Filtering Volume	Measures retrieval specificity by averaging the number of products returned by the generated SQL query, where smaller candidate pools reflect superior structural precision and constraint mapping.	Sec. 5.2.3, 5.2.4
Failure Diagnostics	Quantifies operational stability and error recovery dynamics by measuring average execution attempts (reflecting total pipeline failures) and retry rates (tracking corrections for erroneous SQL, empty returns, or semantic errors).	Sec. 5.2.1, 5.3.3

By systematically calculating these metrics across the `GLM-4.6-355B (Large)`, `GLM-4.5V-106B (Medium)`, and `Qwen-Coder-30B (Small)` models, this evaluation framework provides the explicit empirical data necessary to resolve **RQ2**. Furthermore, by rerunning

this pipeline across various Temperature and Top-p configurations, the methodology systematically isolates how hyperparameter tuning influences retrieval accuracy, system stability, and LLM reasoning within a CRS, which is necessary to answer **RQ3**.

CHAPTER 5

Results and Discussion

Having defined the data generation, user study, complete end-to-end architecture, and their evaluation metrics, the framework is now prepared for empirical testing. This chapter directly applies the previously established methodology to evaluate our system; thus, we present the **empirical contributions** of our research to address our three primary research questions.

We begin in Section 5.1 by detailing the results of the user study to address **RQ1**, providing a quantitative analysis of how effectively an LLM can transform raw behavioural data into natural language queries.

Subsequently, Section 5.2 presents a comparative analysis of the impact of model scale on core performance metrics to address **RQ2**.

Finally, Section 5.3 examines the results pertaining to **RQ3**, isolating the influence of specific LLM hyperparameter configurations on the systemic stability and ranking precision of the CRS framework.

5.1 User Study Results

The methodological setup and consensus framework for this user study are detailed in Section 4.2. In total, 49 participants evaluated 7,054 data-to-question transformations, assessing how effectively the LLM synthesised raw behavioural logs into natural language queries.

Following the predefined data-cleaning audit utilising the golden samples, five participants were excluded from the final analysis. Applying our consensus mechanism (majority vote and conservative tie-breaking) to the remaining valid responses yielded a final evaluation set of 3,450 unique, verified questions.

The inter-rater reliability for this dataset, evaluated using Krippendorff's alpha (α), yielded scores of $\alpha = 0.312$ for representation accuracy, $\alpha = 0.262$ for contextual plausibility, and $\alpha = 0.200$ for the classification of specific generation failure modes. While these values reflect the inherent subjectivity and cognitive complexity of interpreting raw behavioural logs, they further reinforce the necessity of the conservative tie-breaker rule. In the presence of such moderate agreement, adopting the more critical evaluation serves as an important quality-control filter.

5.1.1 Input Characteristics: Session Length and Complexity

Before evaluating the quality of the generated queries, it is essential to understand the structural variations of the input data. The complexity of a user's session directly impacts the cognitive load placed on the LLM during the data-to-text translation phase.

As illustrated in Figure 5.1, the distribution of session lengths (measured by the number of interaction lines in the behavioural logs) is right-skewed. The vast majority of user sessions are relatively brief, falling within the 1-to-5 interaction range, which aligns with the observation that most users engage in quick, focused searches. However, the distribution also exhibits a pronounced "long tail," highlighting highly exploratory sessions that accumulate numerous log lines. As established in Section 4.2, sessions exhibiting multiple behavioural dimensions, such as applying filters while simultaneously exploring specific products, were substantially longer.

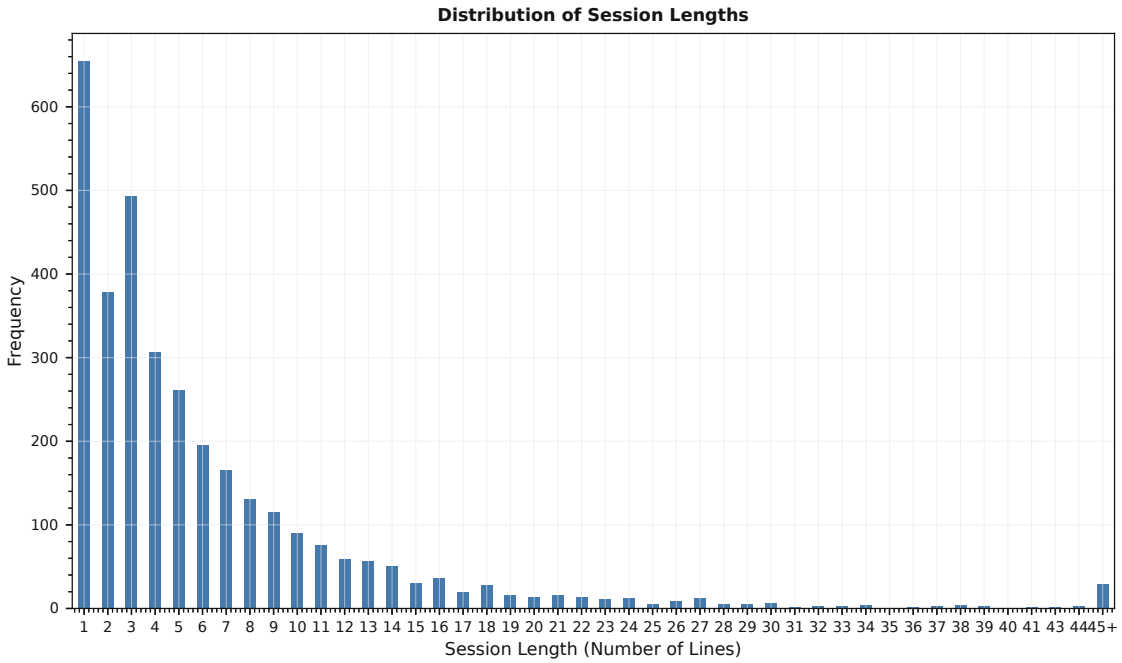


Figure 5.1: Distribution of session lengths measured by the number of log lines.

5.1.2 Output Quality: Overall Representation Accuracy

Figure 5.2 illustrates the absolute counts of "Yes" responses (indicating a successful generation), broken down by the specific behavioural signature of the session. The data reveals that overall accuracy rates generally remain below 40%, underscoring the inherent difficulty of translating raw logs into a cohesive natural language question.

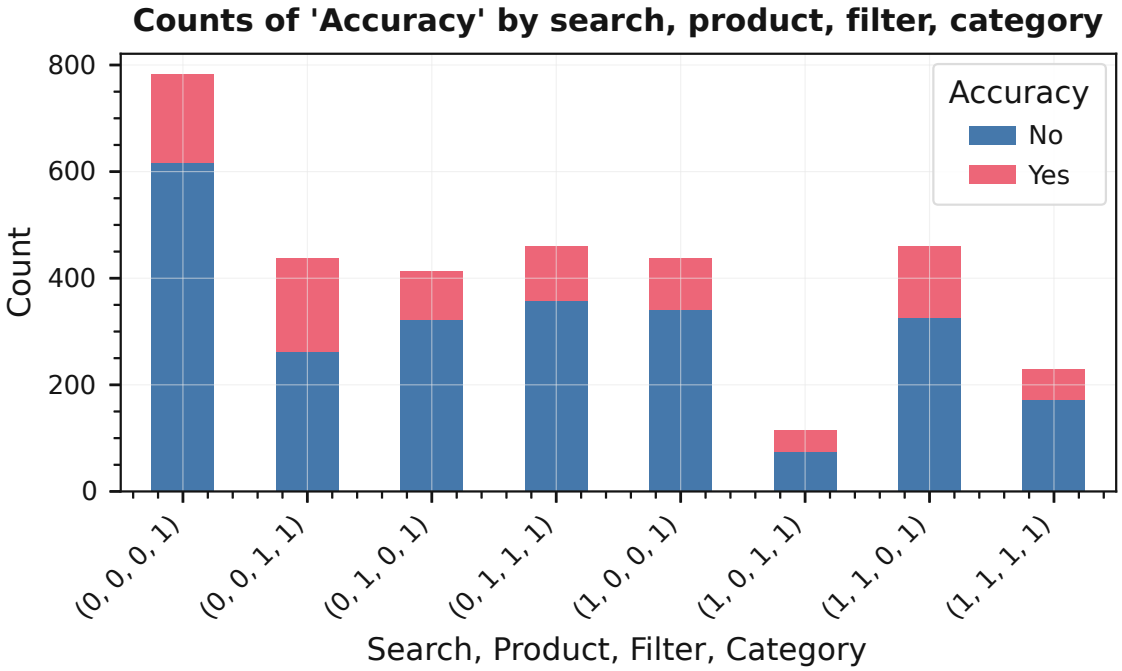


Figure 5.2: Absolute counts of accurately generated queries broken down by behavioural signature.

- **The Grounding Effect of Filters:** The highest-performing session types were `category-filter-search` (∼40%) and `category-filter` (∼35%).
- **The Specificity Penalty (Product Clicks):** A notable drop in accuracy occurs whenever a `product` interaction is introduced into the signature (dropping to ∼20%).
- **Sparse Log Degradation:** The absolute lowest accuracy is observed in sessions consisting exclusively of `category` navigation (∼22%).

5.1.3 Failure Modes: Absolute Frequency of Generation Errors

Given that the representation accuracy consequently dropped below 40%, it is critical to identify the nature of these generative failures.

Figure 5.3 presents the absolute frequencies of these error types. *Hallucination* emerged as the dominant failure mode, indicating that the LLM frequently fabricated constraints,

features, or context that were entirely absent from the input behavioural data. The second most frequent issue was categorised as *not checkable*. Secondary errors, such as *missing entity*, *incorrect entity*, and *incorrect number*, occurred with lower frequency.

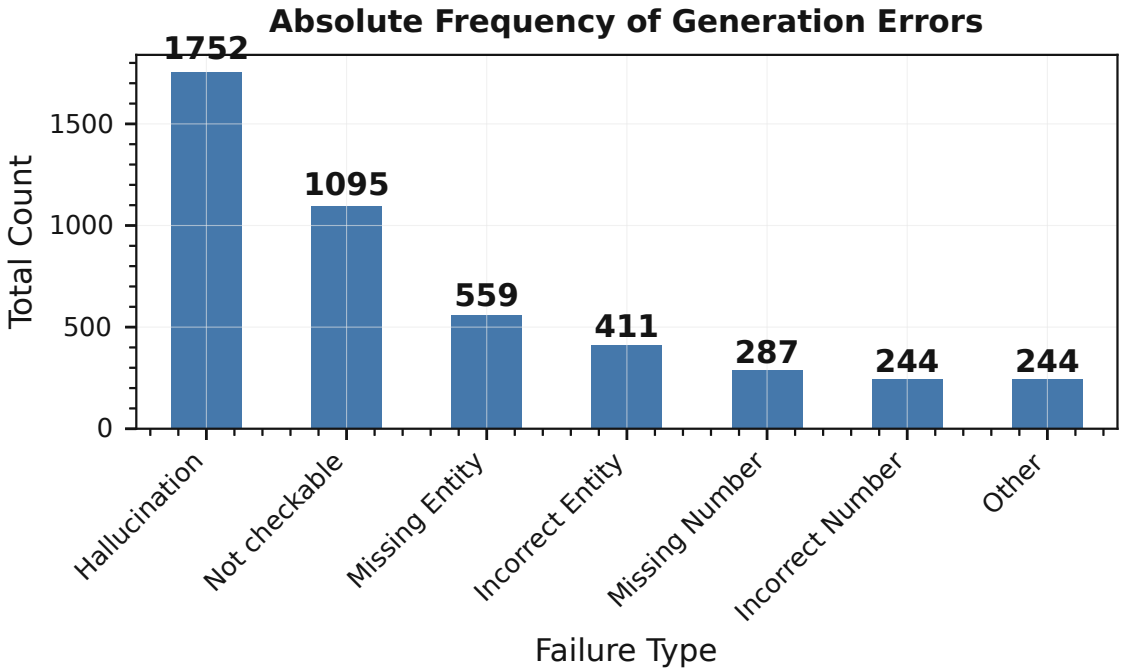


Figure 5.3: Absolute frequency of error types identified during the user study.

5.1.4 Contextual Error Heatmap by Behavioural Signature

To bridge the gap between the overall error frequencies and the varying complexity of the input data, the specific error types were mapped against the established interaction signatures. Figure 5.4 displays the resulting heatmap, revealing precisely what user behaviours trigger specific generation errors:

- **Ambiguity Breeds Hallucination:** In signatures lacking strict parameters, such as category-only sessions (Signature `1, 0, 0, 0`), the hallucination rate peaks dramatically at 70.3%. When missing explicit search terms or numeric filters, the LLM is forced to creatively infer the user's ultimate goal, frequently leading to fabricated constraints in the resulting text.

- **Filters Improve Grounding:** The introduction of explicit filters in the interaction logs improves model grounding. Moving from a generic category exploration to one that includes hard filters (e.g., Signature `1, 1, 0, 0`) provides the LLM with concrete data points to anchor its text generation, thereby reducing the rate of pure hallucination to 37.8%.

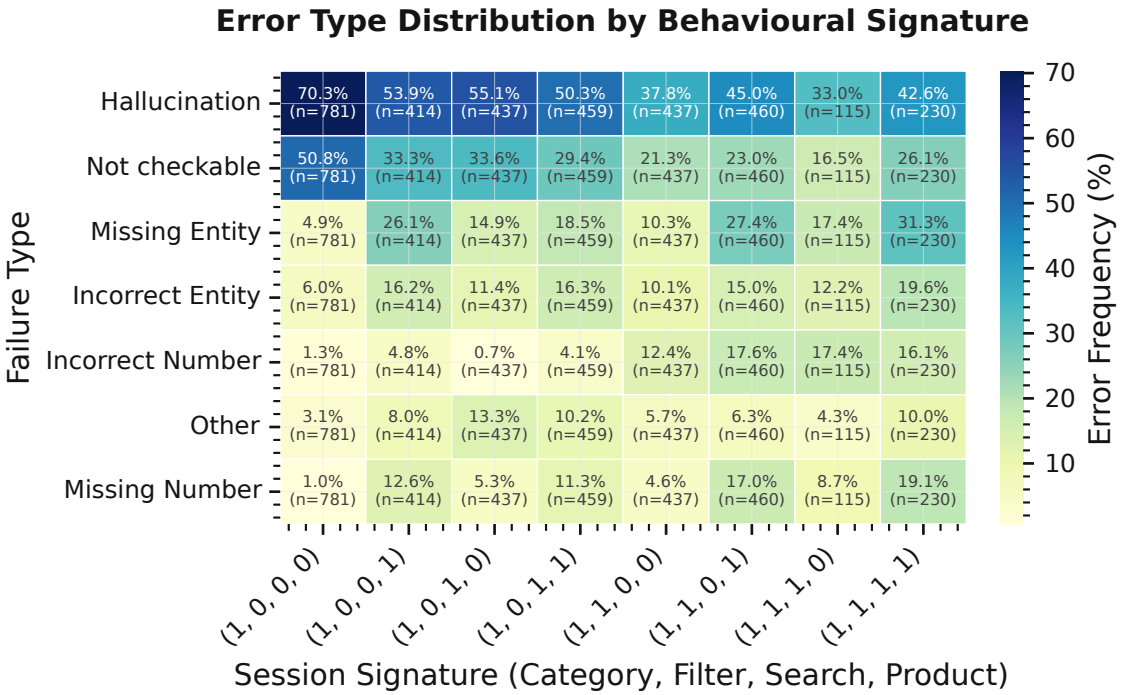


Figure 5.4: Heatmap correlating specific error types with user interaction signatures.

- **The Trade-off of Specificity:** While adding multiple dimensions (such as search and product clicks) helps curb hallucinations, it proportionally elevates extraction errors. As the interaction logs become denser and more technically complex (e.g., Signatures `1, 0, 0, 1` or `1, 1, 1, 1`), errors such as *missing entity* and *missing number* become more prominent. This indicates that while the LLM has more context to work with, it occasionally drops or misinterprets specific numeric values or identifiers when summarising highly complex sessions.

Ultimately, this analysis reveals a fundamental tension in LLM-based data synthesis: sparse logs lead to over-generation and hallucination, while overly dense logs challenge the model's capacity for precise entity extraction and retention.

5.1.5 Contextual Plausibility and Real-World Applicability

As outlined in the methodology, the human evaluation in Label Studio also assessed contextual plausibility (whether the generated query represented a realistic "Real-Use" case that a user might actually ask on the *Geizhals* platform).

Figure 5.5 illustrates the absolute counts of plausible ("Yes") versus implausible ("No") generated queries, segmented by their structural interaction signatures (represented as binary tuples for search, product, filter, and category).

The resulting data reveals a contrast when compared to the strict accuracy metrics discussed previously. Although the LLMs frequently struggled to abstract complex, multi-step behavioural logs into accurate queries, generally achieving an accuracy rate below 40%, they consistently demonstrated the capacity to generate contextually plausible e-commerce questions. Across every distinct behavioural signature, the proportion of positive plausibility assessments ("Yes") outweighed the negative responses ("No"). This indicates that while the models often fail at the exact deterministic translation of raw interaction data, they possess a strong systemic ability to formulate fluent, highly realistic user intents that align with natural shopping behaviour.

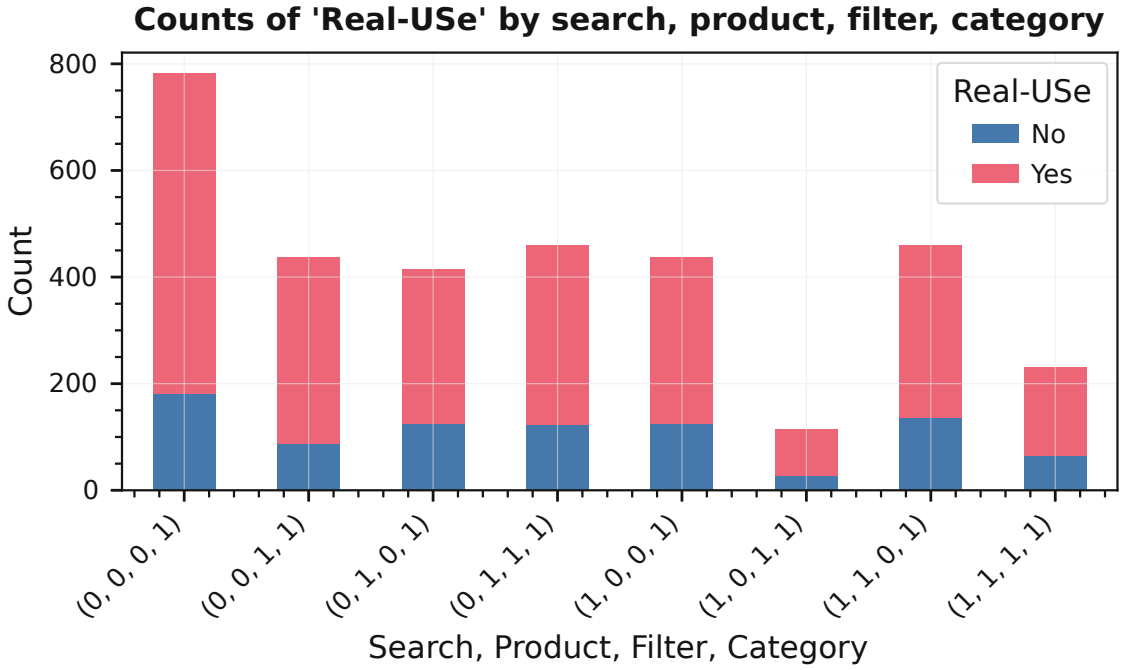


Figure 5.5: Counts of contextually plausible ("Real-Use") generated queries, segmented by interaction signature (search, product, filter, category).

Several key insights can be drawn from this distinction:

- **Plausible Hallucinations in Sparse Logs:** In sessions consisting exclusively of category navigation (Signature `0, 0, 0, 1`), the LLM exhibited its highest absolute volume of plausible queries, despite this same signature previously demonstrating the lowest strict accuracy and highest hallucination rate. This indicates that when the LLM lacks sufficient constraints, it does not generate nonsensical text; rather, it "fills in the blanks" by inventing highly realistic, domain-appropriate e-commerce scenarios that simply do not match the specific ground truth of the log.
- **Domain Knowledge:** Across more complex signatures involving multiple variables, such as queries incorporating filters, searches, and product views (e.g., `1, 1, 1,`

1), the ratio of realistic to unrealistic queries remains overwhelmingly positive. This demonstrates that the Mixtral LLM possesses a strong inherent grasp of consumer hardware terminology and e-commerce conversational tones.

5.1.6 Answer to the Research Question

RQ1: To what extent can an LLM accurately produce question and answer pairs for user simulation in a CRS-context based on behavioural data?

Ultimately, the results indicate that LLMs cannot reliably or accurately produce deterministic, exact-match question and answer pairs that faithfully mirror specific historical user sessions. The human evaluation yielded a predominantly negative result for strict accuracy. The primary catalyst for this failure is the model's inherent propensity to fill informational gaps by injecting unverified constraints. This is evidenced by the overwhelming prevalence of hallucinations and "not checkable" additions.

Conversely, when evaluated purely on its capacity to generate contextually authentic queries typical of a real-world CRS setting, the LLM demonstrated robust performance.

Therefore, to answer the research question: while using LLMs to reverse-engineer exact user intent from raw interaction logs is too prone to factual deviation to be used as a strict ground-truth mapping, the models possess a good understanding of e-commerce domain language. They fail as precise data-to-text translators, but they are exceptionally well-suited as generative engines for creating plausible, synthetic conversational data that mimics real-world consumer inquiries.

5.2 Impact of Different Large Language Models on Recommender Systems

Having established through the user study that the generated queries, despite some deterministic inaccuracies, provide a plausible representation of real-world e-commerce intents (**RQ1**), we now deploy this validated dataset to test the core retrieval capabilities of the CRS. Building on these inputs, the following analysis addresses **RQ2** by comparing how the parameter scale of different LLMs impact their ability to accurately recommend the ground-truth products. For this, we employed the CRS detailed in Section 4.3.2.

The dataset utilised for this comparative evaluation was derived directly from the human-annotated user study. Specifically, we curated a subset by isolating queries that achieved a consensus of "Yes" via majority voting in both the *Accuracy* and *Real-Use* (contextual plausibility) categories. Furthermore, to ensure a fair assessment of the LLMs' retrieval logic, this subset was strictly limited to queries targeting products that could be deterministically resolved within the live database. This final filtering step was necessary to control for the known nomenclature discrepancies between the historical behavioural logs and the active product database schema, ensuring that the models were

only penalised for generation errors rather than structural data mismatches. In total, this left us with 734 unique questions with a ground truth.

5.2.1 Hit Rate and Latency

The initial phase of the evaluation focuses on execution latency and the system Hit Rate, defined as the absolute number of ground-truth products successfully retrieved from the total dataset of 734 valid queries.

As anticipated, retrieval success scaled with model size. The model with the largest parameter count (`GLM-4.6-355B (Large)`) achieved the highest Hit Rate, successfully locating 492 items (see Figure 5.6). Interestingly, the performance gap between the medium and small LLMs was marginal; the 106B and 30B models performed remarkably similarly, retrieving 419 and 413 products, respectively. This suggests that while massive parameter scale provides a distinct advantage in navigating ambiguous queries, specialised coding models (like `Qwen-Coder-30B (Small)`) can punch significantly above their weight class in absolute retrieval tasks.

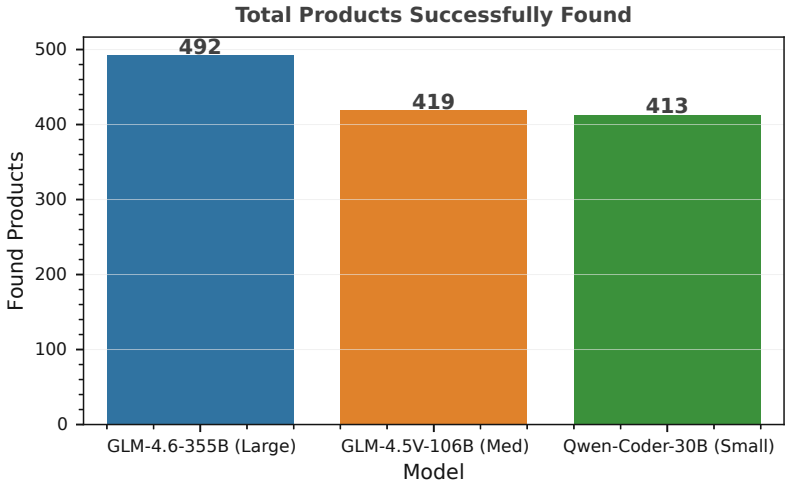


Figure 5.6: Total number of products successfully found by each LLM configuration within the live database.

In addition to retrieval success, system latency represents a critical operational constraint for CRSs. As anticipated, the average execution time scaled with the model's parameter count (see Figure 5.7). The `GLM-4.6-355B (Large)` model exhibited the highest latency, a direct consequence of its massive computational footprint. Furthermore, these extended processing times are inherently linked to the substantial hardware resources (as described in Section 4.3.2) required to host and run such a large model, highlighting the trade-off between the Hit Rate and real-time responsiveness.

Finally, an analysis of the operational failure dynamics, as detailed in Appendix 2, provides insight into the computational effort required by each model scale. The data reveals that

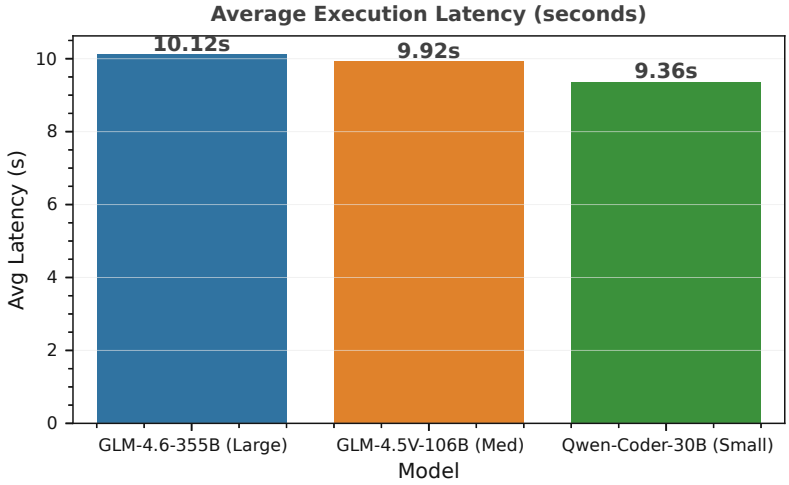


Figure 5.7: Average execution latency (in seconds) across the different model scales and their respective hardware deployments.

the `Qwen-Coder-30B (Small)` model exhibited a significantly higher retry rate and a greater number of average attempts across both successful and unsuccessful queries. This suggests that while the specialised model is a precise retriever, it is more prone to generating initial SQL queries that require the pipeline's autonomous self-correction mechanisms to resolve.

In contrast, the `GLM-4.6-355B (Large)` model demonstrated the highest first-pass reliability, recording the lowest frequency of retries and attempts. This indicates a superior ability to adhere to complex structural constraints without triggering the error-repair loops. The `GLM-4.5V-106B (Medium)` model represents a middle ground in terms of retry frequency; however, it exhibited a disproportionately high average latency during failed retrievals. This peak in execution time suggests that the medium-sized model frequently exhausted the maximum number of recursive retry and relaxation tiers before ultimately failing to produce a valid result.

5.2.2 Ranking Quality Metrics

Building upon the initial analysis of absolute Hit Rates and execution latency, this section evaluates the structural retrieval quality of the system using the ranking metrics established in Section 2.5. To comprehensively assess each model's ability to surface the ground-truth product in an optimal position for the user, performance is analysed across three primary dimensions: MRR, Precision@k, and NDCG@k. A complete numerical breakdown of these performance indicators for all model scales is provided in Table 1 within the Appendix.

For the subsequent analysis of ranking quality, thresholds of $k = 10$, $k = 50$, and $k = 100$ were selected as the primary benchmarks. A failure to include the target product within this range effectively constitutes a retrieval failure. While Precision@k and NDCG@k

are utilised to provide a focused assessment of different interaction windows, the MRR is employed to capture the systemic ranking performance across the entire result set. This provides a singular, weighted measure of how efficiently the system surfaces the ground-truth product relative to the optimal first position.

Furthermore, each metric is presented in two distinct formats:

1. **Total Metric:** Calculated across the entire dataset of 734 queries. Failures to retrieve the product are assigned a score of 0. This provides a measure of *systemic effectiveness*.

2. **Found Metric:** Calculated exclusively for queries where the product was successfully retrieved. This isolates the *ranking quality* of the algorithm, independent of the model's absolute Hit Rate.

MRR

MRR results, illustrated in Figure 5.8, reveal a divergence between systemic success and localised ranking precision across the different model scales.

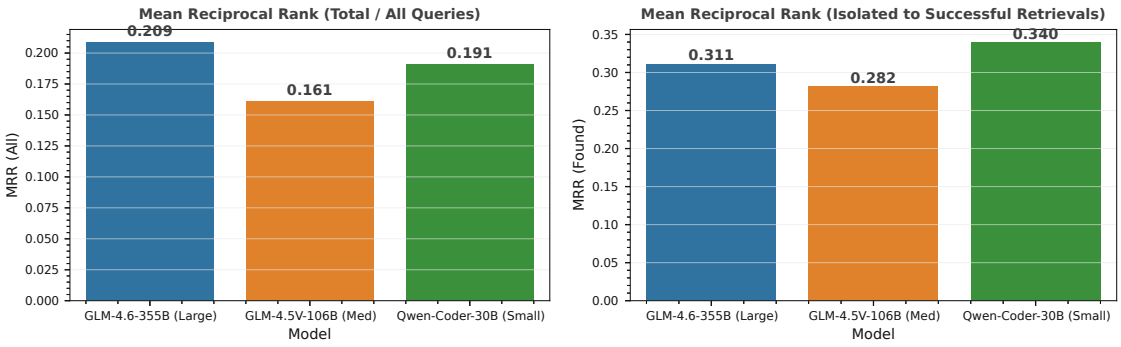
In the *Total MRR* analysis (Figure 5.8a), which accounts for the entire dataset including retrieval failures, the `GLM-4.6-355B (Large)` model achieves the highest score of 0.209. This systemic dominance is largely a byproduct of its superior Hit Rate. Notably, the `Qwen-Coder-30B (Small)` model (0.191) outperforms the `GLM-4.5V-106B (Medium)` (0.161), suggesting that smaller code-specialised models can outperform larger models in retrieval efficacy.

A significant shift in performance dynamics is observed in the *Found MRR* (Figure 5.8b), where the metric is calculated exclusively for successful retrievals. In this isolated view of ranking quality, the `Qwen-Coder-30B (Small)` model achieves the highest score of 0.340, surpassing the larger `GLM-4.6-355B (Large)` (0.311). This suggests that while the smaller, specialised model has a lower absolute success rate, it demonstrates superior ranking quality when the retrieval constraints are successfully identified. Conversely, the `GLM-4.5V-106B (Medium)` model trails in both categories, indicating a potential lack of the structural strictness required for optimal SQL-based ranking within this CRS framework.

Precision@k

The Precision@k results, illustrated in Figure 5.9, demonstrate a trend consistent with the MRR analysis, further clarifying the models' operational efficacy across various recommendation window sizes.

As shown in the *Total Precision@10* results (Figure 5.9a), the systemic performance is led by the `GLM-4.6-355B (Large)` model with a score of 0.377. This result is intrinsically linked to the model's superior absolute Hit Rate, as the metric penalises any



(a) Total MRR. (b) MRR isolated to successful retrievals.

Figure 5.8: Comparison of Total vs. Found Mean Reciprocal Rank.

retrieval failure or placement outside the top ten. The `Qwen-Coder-30B (Small)` model maintains its position as the second-most effective system (0.337), outperforming the `GLM-4.5V-106B (Medium)` architecture (0.293). These performance tiers remain stable as the evaluation window expands; for *Total Precision@100*, the Large model reaches 0.589, followed by the Small model at 0.507.

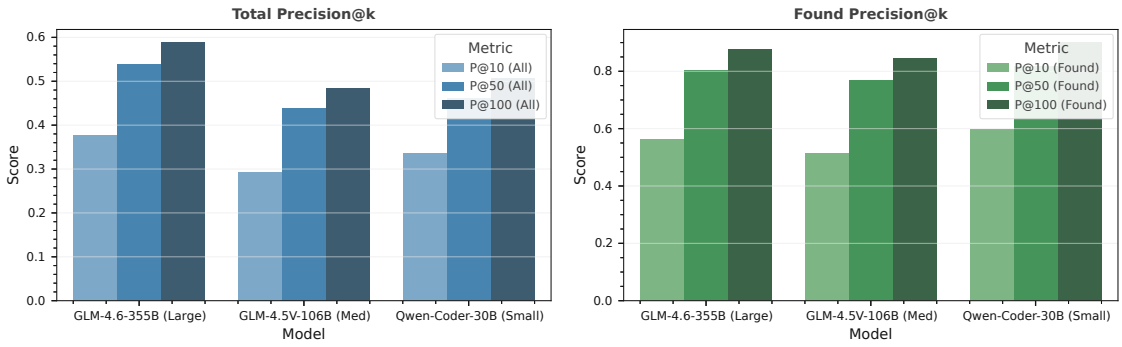
The results for *Found Precision@k* (Figure 5.9b) highlight a critical shift in localised ranking accuracy. When isolated to successful retrievals, the `Qwen-Coder-30B (Small)` model achieves the highest precision across all evaluated thresholds. At $k = 10$, it achieves a score of 0.598, surpassing both the Large (0.563) and medium (0.513) models. This trend is amplified at larger cut-offs: for *Found Precision@100*, the Small model achieves a score of 0.901, indicating that in over 90% of successful retrievals, the target product is located within the first 100 results.

This consistency across $k = 10, 50$, and 100 confirms that while the smaller, specialised model has a lower absolute retrieval volume, it is remarkably efficient at surfacing identified products within the relevant recommendation slots. These findings reinforce the conclusion that the specialised code-generation model provides a high degree of structural precision in SQL-based retrieval tasks, effectively surfacing the ground-truth product more reliably than larger, general-purpose models.

NDCG@k

NDCG results, illustrated in Figure 5.10, offer a granular evaluation of ranking quality by accounting for the specific position of the ground-truth product within the recommendation list.

As shown in the *Total NDCG@10* results (Figure 5.10a), the `GLM-4.6-355B (Large)` model demonstrates the highest systemic performance with a score of 0.241. This lead is sustained by the model's overall retrieval reliability. The `Qwen-Coder-30B (Small)` model (0.220) continues to exhibit a stronger systemic performance than the `GLM-4.5V-106B (Medium)` model (0.186). Consistent with the patterns observed in

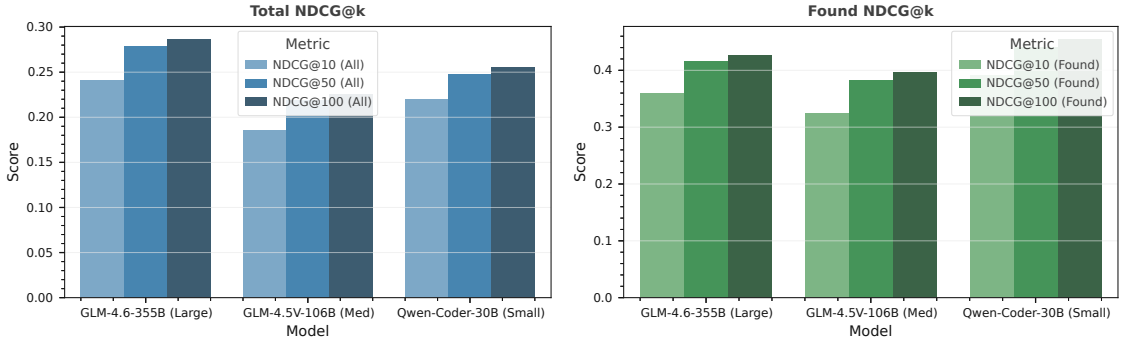


(a) Total Precision@k across all queries. (b) Precision@k isolated to successful retrievals.

Figure 5.9: Comparison of Total vs. Found Precision@k.

the Precision and MRR analyses, these performance tiers remain stable across the higher thresholds of $k = 50$ and $k = 100$.

The *Found NDCG@k* results (Figure 5.10b) further validate the superior localised ranking precision of the smaller, specialised architecture. When isolated to successful retrievals, the `Qwen-Coder-30B (Small)` model achieves an NDCG@10 of 0.391, surpassing the Large model's score of 0.360. This trend is maintained across the full evaluation range, with the Small model achieving a Found NDCG@100 of 0.454. The higher gain scores suggest that the code-generation model is more effective at formulating SQL logic that adds more constraints than the other models.



(a) Total NDCG@k performance demonstrating the overall ranking quality. (b) NDCG@k performance for successfully retrieved products.

Figure 5.10: Comparison of Total vs. Found NDCG@k.

5.2.3 Retrieval Volume and Constraints Analysis

Beyond ranking quality, a critical dimension of a CRS is its retrieval specificity, the ability to utilise contextual information to effectively narrow the search space. A smaller, highly targeted result set minimises the users' cognitive load. To assess this, we evaluated the average number of products retrieved per query and analysed how varying levels of

information density (session length and interaction signatures) influenced the models' filtering behaviour.

An analysis of the global average output volume (Figure 5.11) reveals a clear hierarchy in filtering stringency. Among successful executions, the `GLM-4.5V-106B (Medium)` model returned the largest candidate pools, averaging 743.9 products per query. The `GLM-4.6-355B (Large)` model followed with 638.5 items, while the `Qwen-Coder-30B (Small)` model retrieved significantly fewer, averaging only 494.7 products. This confirms our previous observations regarding the 30B model, as it consistently favours restrictive SQL clauses over broad, safe category retrievals.

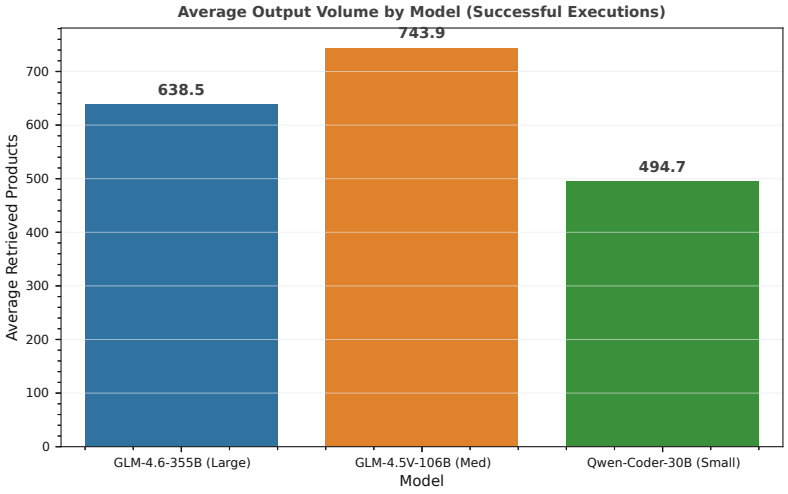


Figure 5.11: Average output volume (number of retrieved products) mapped by LLM scale during successful executions.

Breaking down the retrieval volume by behavioural signature (Figure 5.12) illustrates how explicit constraints guide the LLMs. Across almost all signatures, queries that contain multiple behavioural dimensions, particularly those combining explicit `Filter` actions with `Search` strings (e.g., signatures `1110` or `1111`), yield substantially smaller, more refined result sets.

Furthermore, the data highlights distinct model behaviours when handling filters. The Medium model tends to apply the least restrictive filtering logic, often returning broader results even when explicit filter interactions are present in the log. In contrast, the Small model consistently outputs the most constrained pools whenever a filter is applied, underscoring its superior structural precision when mapping natural language constraints to exact SQL operators.

Finally, examining the impact of raw session length (Figure 5.13) reveals a distinct negative correlation between the amount of contextual interaction data and the final output volume. For both the `GLM-4.6-355B (Large)` ($r = -0.1455$) and `Qwen-Coder-30B (Small)` ($r = -0.1486$) models, as the session length increases, the average number of retrieved products drops noticeably. This indicates that longer, exploratory sessions

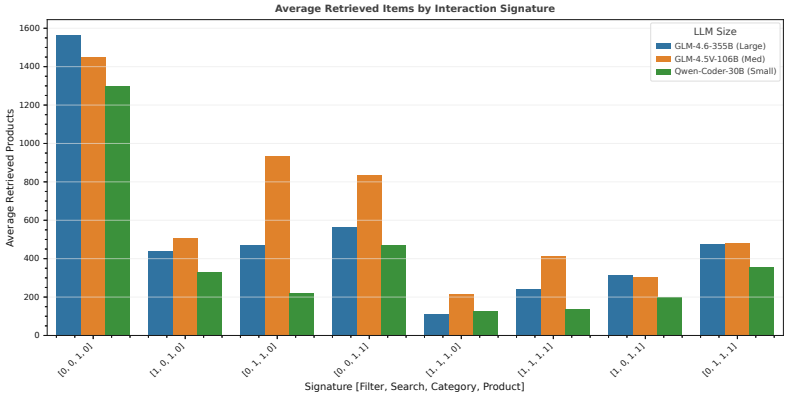


Figure 5.12: Average retrieved items categorised by user interaction signature [Filter, Search, Category, Product].

provide the LLMs with a higher density of semantic constraints, allowing them to construct highly restrictive `WHERE` clauses.

However, a notable anomaly occurs with the `GLM-4.5V-106B (Medium)` model ($r = -0.1141$), which exhibits a sharp spike in retrieval volume for long sessions (21+ lines), slightly weakening its overall negative correlation. This deviation can be explained by referencing the system's operational fallback mechanisms (detailed in Table 2). Confronted with a higher level of complexity inherent in highly elongated sessions, the Medium model struggles to produce a valid, tightly constrained query. Consequently, it triggers the pipeline's *Semantic Relaxation* loop more frequently. By aggressively stripping away failing `WHERE` conditions to avoid a "zero-row" error, the fallback mechanism inherently floods the final result set, leading to the observed volume spike.

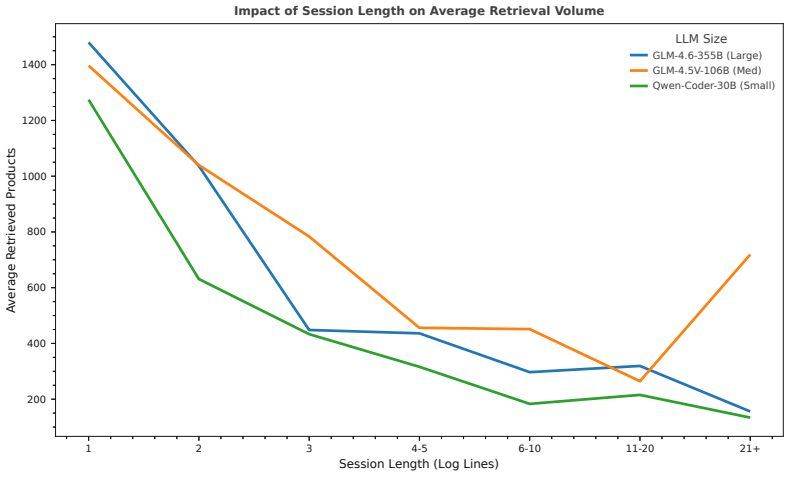


Figure 5.13: Impact of the original session length (log lines) on the resulting average retrieval volume.

5.2.4 Additional Head-to-Head Comparisons

To further isolate the operational nuances of the different LLMs, we conducted two additional head-to-head comparisons using specific subsets of the data: an *intersection subset* (queries where the target product was successfully found by all three models) and a *filter-active subset* (queries derived from sessions containing explicit filter interactions).

By isolating the 320 queries successfully resolved by all models, we eliminate the statistical skew caused by Hit Rate disparities (Figure 5.14). Within this subset, the established performance trends remain visible, albeit less pronounced. The `Qwen-Coder-30B (Small)` model maintains its position as the most precise ranker, achieving the highest MRR (0.335) and consistently returning the fewest average items.

Crucially, in this "success-only" subset, the previously observed anomaly for the `GLM-4.5V-106B (Medium)` model concerning highly elongated sessions (21+ lines) completely disappears; the retrieval volume trends downwards in unison with the other models. This confirms our earlier hypothesis: the Medium model's volume spike on long sessions is directly tied to its failure rate. When the Medium model successfully parses a complex query on the first attempt, it filters normally. The spikes only occur when it fails and triggers the pipeline's semantic relaxation fallback. Additionally, execution latency across all three models normalises to approximately 8.9 seconds, further proving that the latency disparities observed in the global dataset are driven primarily by error-recovery loops on failed queries.

The second subset (Figure 5.15) examines queries generated from sessions where users explicitly applied filters, which inherently provide the LLM with hard constraints (e.g., specific brands or price limits). In this scenario, the `GLM-4.6-355B (Large)` model demonstrates clear systemic dominance, achieving a 59.7% Hit Rate and the highest global MRR (0.189). Its massive parameter scale equips it to better interpret and map these explicit natural language constraints to the database schema without triggering errors.

However, the underlying trend of filtering stringency remains consistent: despite the Large model achieving higher overall success, the Small model still retrieves significantly fewer items on average (241.8 vs. 337.7). Furthermore, examining the retrieval volume by session length within this filter-heavy subset reveals the recurrence of the Medium model's anomaly for extended sessions (21+ lines). This indicates that long user sessions densely packed with explicit filters are the specific catalyst that breaks the Medium model's logic, forcing it into the semantic relaxation loop and subsequently flooding the result set. Conversely, the Small model's strict filtering logic introduces its own operational trade-off. Its propensity to generate highly restrictive SQL queries frequently results in over-constrained conditions that return empty candidate lists (zero rows). As detailed in the system execution data (Table 2), this stringency directly translates into a significantly higher overall retry rate for the Small model (2.45%) compared to the Medium (0.82%) and Large (0.27%) models, as the pipeline attempts to salvage the zero-row returns.

Figure 5.14: Direct comparison on only sessions where the target item was found by all LLMs.

5.2.5 Answer to the Research Question

RQ2: To what extent does the size of an LLM impact the performance of a CRS?

Based on the empirical evaluation, we conclude that while the parameter scale of an LLM significantly impacts the systemic reliability of a CRS, it is not the sole, nor always the primary, determinant of retrieval quality. The impact of model size is fundamentally split between absolute retrieval volume, ranking precision, and resource requirements:

- **Systemic Reliability and Hit Rate:** Model scale dictates the absolute success rate. The massive `GLM-4.6-355B (Large)` model demonstrated a superior capacity to interpret ambiguous natural language constraints, resulting in the highest overall Hit Rate and the lowest frequency of autonomous retries. It serves as the most resilient foundation for general user queries.

Figure 5.15: Direct comparison on only sessions where a filter was used.

- **Specialisation over Scale for Precision:** Conversely, parameter count does not equate to ranking accuracy. The specialised `Qwen-Coder-30B (Small)` model proved that highly targeted training (specifically in code and SQL generation) allows the smaller model to significantly outperform massive models in structural precision. When successful, the specialised model acts with high filtering stringency, retrieving smaller candidate pools and achieving consistently superior localised ranking metrics (Found MRR, Precision@k, and NDCG@k).
- **Operational Trade-offs:** The size of the model inherently dictates the operational constraints of the system. The 355B model requires vast hardware resources and imposes a high baseline latency, challenging the real-time requirements of a CRS. While the 30B model generates tokens much faster, its slightly higher propensity for initial structural errors means it relies more heavily on the pipeline's autonomous self-correction loops to function properly.

Therefore, to answer the research question: model scale heavily dictates a system's

absolute capacity to locate products and minimises error-handling overhead. However, architectural specialisation, such as training tailored to code generation, is also a stronger predictor of filtering stringency and exact ranking precision than raw parameter count.

5.3 Impact of Different Hyperparameters on Recommender Systems

While the previous analysis demonstrates that architectural scale primarily governs absolute Hit Rates and ranking precision, this section addresses **RQ3** by isolating how specific hyperparameter tunings further influence this established baseline. Specifically, we conducted a grid search varying the Temperature ($T \in \{0.25, 0.50, 0.75\}$) and Top-$p$ ($p \in \{0.7, 0.8, 0.9\}$) settings. To ensure that observed performance variances were attributable solely to parameter modulation rather than architectural differences, the evaluation was performed exclusively using the `GLM-4.6-355B (Large)` model, which demonstrated the highest systemic performance across all primary metrics in the preceding benchmarks.

These specific hyperparameters were selected because they act as the primary mechanisms for modulating the determinism and stochasticity of the LLM's generative output. *Temperature* adjusts the underlying logits before sampling, effectively controlling the entropy or "creativity" of the model; lower values yield more focused, predictable responses, while higher values increase randomness. *Top-p* (also known as nucleus sampling) dynamically restricts the candidate token pool during generation to the smallest set of tokens whose cumulative probability exceeds the threshold p, thereby truncating the "long tail" of unlikely outputs. By tuning these parameters, we aim to find the optimal balance between strict structural adherence (required for SQL syntax) and sufficient semantic flexibility (required to interpret diverse user queries) [Aro+24]. This specific grid subset was chosen to capture the most impactful shifts in model behaviour while remaining feasible within our computational resource constraints.

The full systemic results of this grid search are documented in Table 3, and a visual breakdown of the performance distributions is provided in Figure 5.16.

5.3.1 Operational Dynamics and Latency

In addition to minor shifts in ranking precision, the hyperparameter configurations directly influenced the operational speed of the CRS. As illustrated in the latency analysis (Figure 5.16), configurations favouring deterministic behaviour (lower Temperature and lower Top-p) generally resulted in faster average execution times. This operational efficiency is likely driven by two factors: the model selects high-probability structural tokens more rapidly, and the tighter constraints lower the probability of generating hallucinated columns or syntax errors, thereby bypassing the pipeline's time-consuming autonomous error-repair loops.

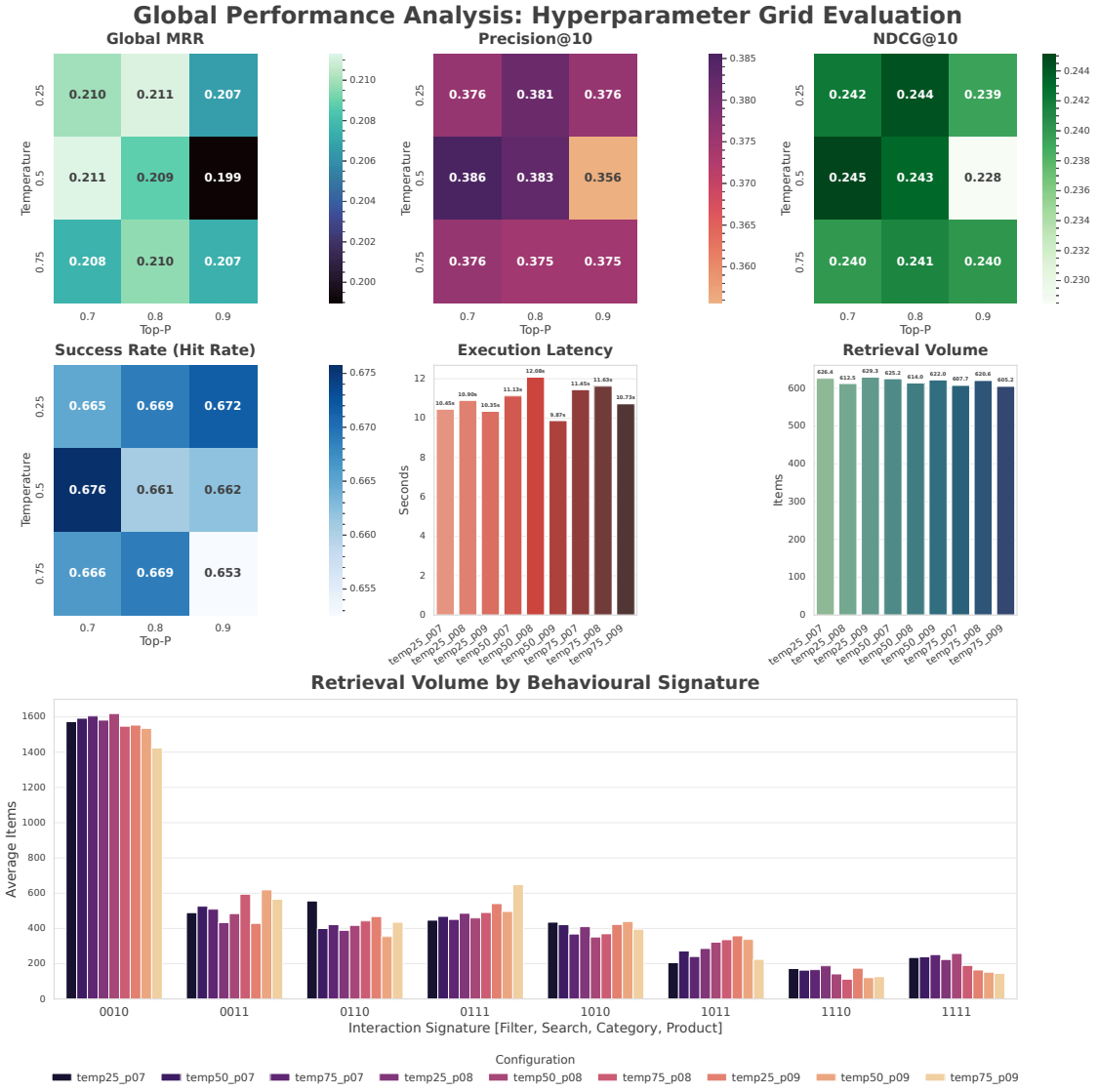


Figure 5.16: Dashboard on all metrics regarding the different hyperparameter settings.

5.3.2 Retrieval Consistency and Ranking Quality

Across the evaluated grid, the fundamental retrieval capabilities of the CRS remained highly stable. The total number of successfully retrieved products fluctuated within a narrow band of 479 to 496 items, representing absolute Hit Rates between 65.3% and 67.6%. This consistency suggests that for highly structured tasks like SQL generation, the core prompt architecture and the pipeline's deterministic fallback mechanisms exert a much stronger influence on systemic success than granular shifts in sampling probabilities.

While the absolute Hit Rates remained stable, the ranking quality metrics exhibited slight sensitivities to the configuration state. Generally, settings that enforced more deterministic token selection, specifically those with lower Top-p values ($p = 0.7$ and $p = 0.8$), yielded slightly superior ranking performance. The highest systemic MRR (0.211), Precision@10 (0.386), and NDCG@10 (0.245) were observed at the $T = 0.50, p = 0.7$ configuration. A similar behaviour was noted for higher k thresholds.

Conversely, expanding the probability mass too widely introduced minor semantic drift. A notable negative outlier occurred at $T = 0.50, p = 0.9$, which scored the lowest across systemic MRR (0.199), Precision@10 (0.356), and NDCG@10 (0.228). This indicates that while higher Top-p values allow for more diverse natural language responses, they occasionally cause the LLM to select less optimal, heavily generalised SQL filters, thereby slightly degrading the exactness of the ranking.

5.3.3 Error and Retry Analysis

To further contextualise the operational impact of these hyperparameter configurations, we conducted an analysis of the system's execution dynamics, specifically examining the average attempts, retry rates, and overall generation error rates.

As illustrated in Figure 5.17, the average number of execution attempts per query remains tightly clustered, though distinct patterns emerge. During our evaluation, configurations favouring deterministic behaviour led to fewer average attempts. While there are isolated outliers, the data indicates that more stochastic settings force the pipeline to engage its self-correction loops more frequently.

This observation is corroborated by the retry rate data (Figure 5.18), which perfectly mirrors the trend seen in the average attempts. It is important to note that the retry rate in this specific context refers exclusively to instances where the initial SQL execution returned an empty candidate list (zero rows), thereby triggering the semantic relaxation fallback. Lower Temperature and Top-p values correlate with a decreased reliance on this mechanism, demonstrating that restricting the LLM's "creativity" yields more accurately constrained initial queries that successfully map to available inventory, rather than generating overly specific filters that result in zero matches. Interestingly, this established pattern is interrupted only by a distinct outlier at the highest level of stochasticity.

Finally, Figure 5.19 highlights the explicit generation error rates. While a peak error rate of approximately 3% is observed in the top-right quadrant (representing a specific

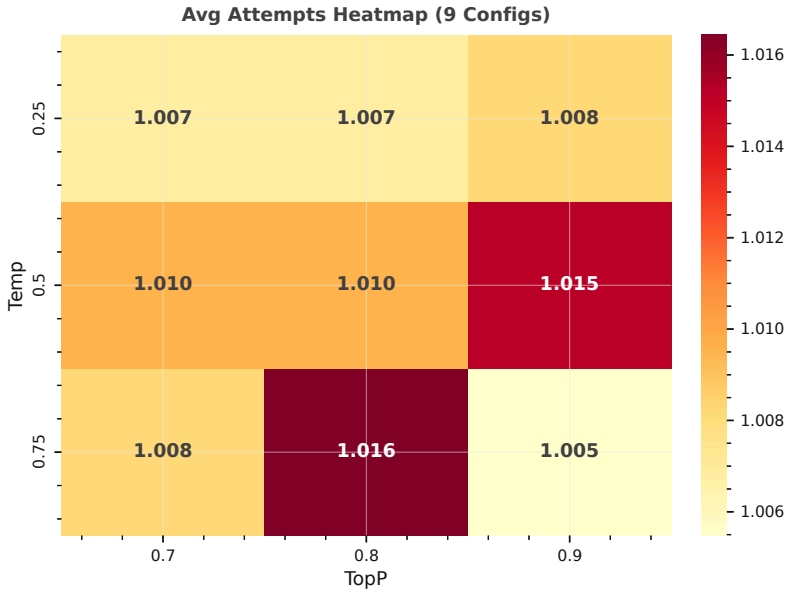


Figure 5.17: Heatmap illustrating the average number of execution attempts across different Temperature and Top-p configurations.

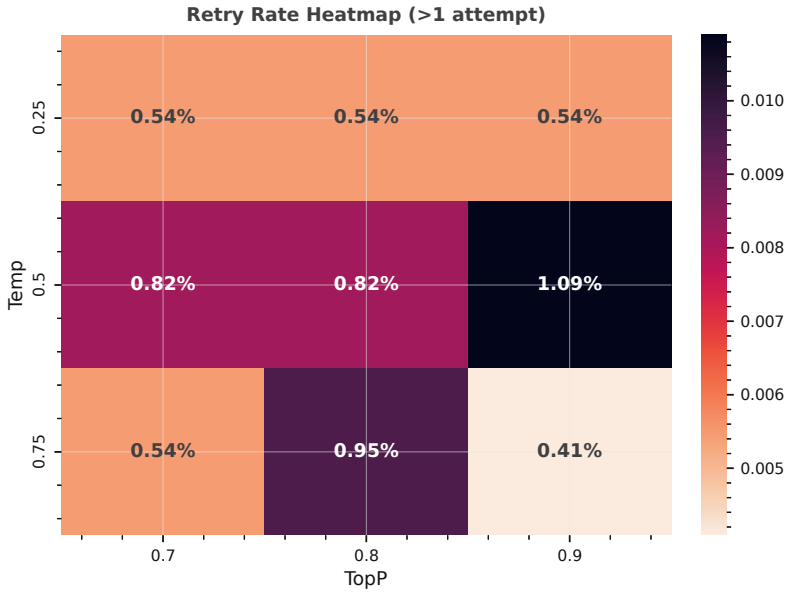


Figure 5.18: Heatmap of the system retry rates across the evaluated hyperparameter grid.

convergence of high probability mass and lower Temperature), the broader distribution confirms the central hypothesis: more deterministic configurations are inherently less error-prone. By preventing the model from sampling "long-tail" tokens, the system naturally avoids hallucinating invalid database columns or breaking strict SQL syntax.

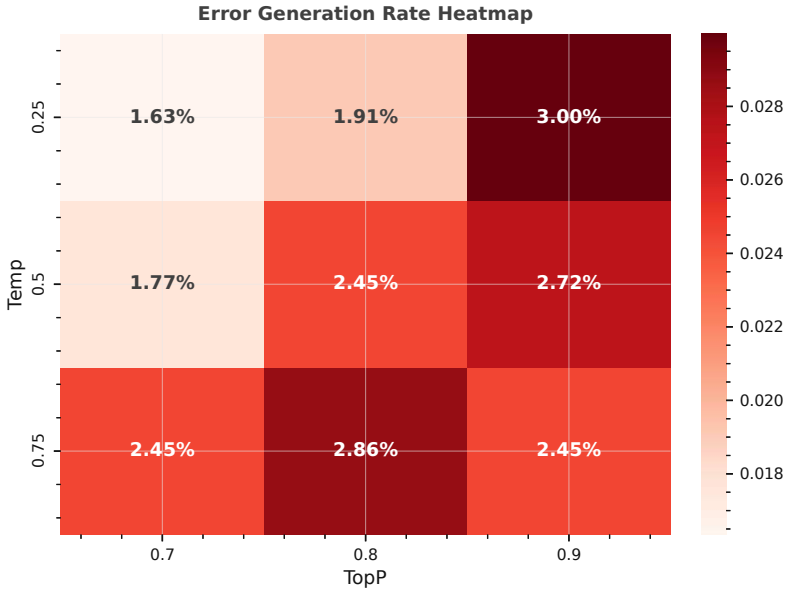


Figure 5.19: Heatmap demonstrating the overall generation error rate of the different configurations.

5.3.4 Answer to the Research Question

RQ3: To what extent do various hyperparameters and configurations of LLMs influence the performance of a CRS? Based on the empirical grid evaluation, we conclude that hyperparameter configurations exert only a marginal, fine-tuning influence on the overall performance of a pipeline-driven CRS. Because the architecture relies on strict SQL generation rather than open-ended conversational text, the stochastic "creativity" modulated by these parameters is largely superseded by the system's deterministic prompt structure and fallback mechanisms.

Specifically, we observe the following dynamics:

- **Systemic Stability:** The absolute retrieval capacity remains highly stable across all evaluated settings, successfully finding between 479 and 496 items regardless of configuration.
- **The Advantage of Determinism:** Configurations that enforce more deterministic token selection (particularly lower Top-p values like $p = 0.7$) provide measurable operational advantages. They yield slightly superior ranking quality (higher MRR,

Precision@10, and NDCG@10) and faster average execution latencies due to a reduction in syntax errors and autonomous retries.

- **The Cost of Stochasticity:** Expanding the probability mass (e.g., $p = 0.9$) occasionally introduces semantic drift, leading to less optimal query filtering and an increased reliance on the system's error-recovery loops.

Therefore, while hyperparameter optimisation facilitates slight trade-offs between execution speed and localised ranking exactness, there are no profound systemic deviations across the grid. The stability of these results ultimately proves that a robust architectural pipeline is vastly more critical to CRS success than granular parameter tuning.

CHAPTER 6

Conclusion

The empirical findings presented in the previous chapters highlight both the generative capabilities and the strict operational boundaries of LLMs within our designed CRS framework. To synthesise these insights and contextualise their broader impact on the future development of CRSs, this final chapter evaluates the overarching contributions of the thesis.

Section 6.1 outlines the core key takeaways derived from our empirical findings and methodological artifacts. Section 6.2 acknowledges the inherent boundaries of the experimental design and directly links these limitations to concrete, promising directions for future research.

6.1 Key Takeaways and Contributions

The primary objective of this thesis was to systematically evaluate the impact of LLM scale and hyperparameter configurations on the performance of a CRS, grounded in real-world user behaviour. By addressing the central research questions, this work delivers three concrete scientific contributions:

1. Methodological Contribution

This research bridges the semantic gap between implicit user behaviour and explicit conversational queries. We developed a novel, reproducible pipeline for data synthesis and human validation, yielding a benchmark dataset of 734 interaction-to-query pairs derived directly from real-world e-commerce clickstreams.

2. Technical Contribution

We designed and implemented a resilient, modular end-to-end CRS evaluation framework. The architecture moves beyond simple zero-shot prompting by integrating a dynamic SQL-

generation pipeline equipped with autonomous syntax-repair and semantic-relaxation loops, providing a reliable testbed for benchmarking varying LLMs, hyperparameter configurations, and re-ranking algorithms.

3. Empirical Contributions

Through the application of this technical framework, the thesis provides empirical insights into the operational dynamics of LLMs within RSs:

- **The Semantic Gap in Intent Translation:** While LLMs excel at generating plausible, domain-appropriate e-commerce dialogue, they struggle as deterministic translators of raw clickstream logs. Faced with sparse data, they exhibit a strong propensity to hallucinate constraints, proving that reverse-engineering exact user intent requires more than zero-shot reasoning.
- **Size is Not the Only Determining Factor:** In active product retrieval, raw parameter scale (e.g., a 355B model) drives absolute systemic reliability and hit rates. However, architectural specialisation can trump size for exact ranking quality. The code-specialised model (`Qwen-Coder-30B (Small)`) exhibited superior structural stringency, mapping constraints to SQL with higher precision, with the trade-off of finding fewer products.
- **Limited Impact of Different Hyperparameters:** An extensive grid search revealed that Temperature and Top-p configurations exert only a marginal, fine-tuning influence. The overall success and operational latency of a CRS are dictated far more by deterministic prompt engineering, schema refinement, and autonomous fallback mechanisms than by granular shifts in token sampling probabilities.

6.2 Limitations and Directions for Future Research

While this thesis provides a comprehensive comparative analysis, several inherent limitations exist within the data and computational scope. To advance the field of CRSs, these specific boundaries directly inform the following avenues for future research:

- **Limitation: Generative Hallucinations in Data Synthesis**
  → Future Work: Advanced Behavioural Translation Workflows
  The user study revealed that LLMs have a tendency to hallucinate constraints when translating sparse historical logs to natural language questions. To improve synthetic dataset generation, future research must optimise this translation process. Investigating advanced agentic workflows, Chain-of-Thought prompting, or specialised embedding models could better bridge the semantic gap between implicit clickstream data and explicit natural language without introducing fabricated context.

- **Limitation: Computational Constraints of Massive Models**
  → Future Work: Supervised Fine-Tuning on Smaller Architectures
  The sheer operational scale of deploying massive models (up to 355 billion parameters) imposed severe computational overhead and latency, restricting real-time viability. Conversely, smaller models were faster but required frequent reliance on autonomous error-correction loops. Future work can utilise the output of the larger model to conduct supervised fine-tuning on smaller, open-weights LLMs (e.g., 8B to 14B parameters). By explicitly training them on successful CRS interaction traces, researchers could potentially achieve the high retrieval reliability of a 355B model with the low latency required for live production environments.

- **Limitation: Static, Single-Turn Evaluation**
  → Future Work: Multi-Turn Conversational Dynamics
  The current framework and curated dataset evaluate retrieval performance based on single, zero-shot synthesised queries. While this isolates absolute retrieval accuracy, real-world CRSs are inherently dialogue-driven. Future research should extend this evaluation pipeline to support long-context, multi-turn conversations. This would allow researchers to measure an LLM's capacity to maintain context history, dynamically refine SQL constraints, and intelligently recover from direct user corrections over extended, exploratory shopping sessions.

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence. 13, 41, 75, 106

CRS Conversational Recommender System. 1–5, 7, 9, 10, 13, 14, 16, 17, 21, 29, 30, 34–36, 38, 39, 44, 45, 51, 52, 54, 56, 60–62, 64, 66, 67, 69–71, 73, 75

LLM Large Language Model. 2–5, 7, 11, 13–18, 21, 29–33, 35–52, 57–60, 62, 64, 66, 69–71, 73–75

MoE Mixture-of-Experts. 12, 33, 75

MRR Mean Reciprocal Rank. 4, 18, 43, 53–56, 59, 61, 64, 66, 75, 95, 96

NDCG Normalised Discounted Cumulative Gain. 4, 17, 18, 43, 53, 55, 56, 61, 64, 67, 73, 75, 95, 96

NLP Natural Language Processing. 2, 7, 18, 75

RS Recommender System. 1, 3, 7–9, 13, 70, 75

Bibliography

[Ali25] Alibaba Cloud. *Qwen-Coder-30B*. https://huggingface.co/Qwen/Qwen3-Coder-30B-A3B-Instruct. 2025.

[Ani+23] Rohan Anil et al. „Palm 2 technical report“. In: *arXiv preprint arXiv:2305.10403* (2023).

[AP25] Yadagiri Annepaka and Partha Pakray. „Large language models: a survey of their development, capabilities, and applications“. In: *Knowledge and Information Systems* 67.3 (2025), pp. 2967–3022.

[Aro+24] Chetan Arora et al. „Optimizing LLMs for Code Generation: Which Hyperparameter Settings Yield the Best Results?“ In: *2024 31st Asia-Pacific Software Engineering Conference (APSEC)*. 2024, pp. 281–290. DOI: 10.1109/APSEC65559.2024.00039.

[Bac26] Nidhal Baccouri. *deep-translator*. https://pypi.org/project/deep-translator/. 2026.

[BR20] D. Bawden and L. Robinson. „Information Overload: An Overview“. In: *Oxford Encyclopedia of Political Decision Making*. This is a draft of a chapter that has been published by Oxford University Press in the book Oxford Encyclopedia of Political Decision Making. Oxford: Oxford University Press, June 2020. DOI: 10.1093/acrefore/9780190228637.013.1360. URL: https://openaccess.city.ac.uk/id/eprint/23544/.

[Bob+13] Jesús Bobadilla et al. „Recommender systems survey“. In: *Knowledge-based systems* 46 (2013), pp. 109–132.

[Bur+00] Robin Burke et al. „Knowledge-based recommender systems“. In: *Encyclopedia of library and information systems* 69.Supplement 32 (2000), pp. 175–186.

[Bur07] Robin Burke. „Hybrid web recommender systems“. In: *The adaptive web: methods and strategies of web personalization* (2007), pp. 377–408.

[Cha22] Harrison Chase. *LangChain*. Oct. 2022. URL: https://github.com/langchain-ai/langchain.

[Chr25] Chroma. *Chroma: AI-native open-source embedding database*. 2025. URL: https://www.trychroma.com.

[Dev+19] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[Di 23] Dario Di Palma. „Retrieval-Augmented Recommender System: Enhancing Recommender Systems with Large Language Models“. In: *Proceedings of the 17th ACM Conference on Recommender Systems*. RecSys '23. Singapore, Singapore: Association for Computing Machinery, 2023, pp. 1369–1373. ISBN: 9798400702419. DOI: 10.1145/3604915.3608889. URL: https://doi.org/10.1145/3604915.3608889.

[Fan+23] Wenqi Fan et al. *Recommender Systems in the Era of Large Language Models (LLMs)*. 2023. arXiv: 2307.02046 [cs.IR].

[FTD25] Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. „Reasoning beyond limits: Advances and open problems for LLMs“. In: *ICT Express* 11.6 (Dec. 2025), pp. 1054–1096. ISSN: 2405-9595. DOI: 10.1016/j.icte.2025.09.003. URL: http://dx.doi.org/10.1016/j.icte.2025.09.003.

[Fri+23] Luke Friedman et al. „Leveraging Large Language Models in Conversational Recommender Systems“. In: *arXiv preprint arXiv:2305.07961* (2023).

[GCS23] Sebastian Gehrmann, Elizabeth Clark, and Thibault Sellam. „Repairing the cracked foundation: A survey of obstacles in evaluation practices for generated text“. In: *Journal of Artificial Intelligence Research* 77 (2023), pp. 103–166.

[GJ17] Jyotirmoy Gope and Sanjay Kumar Jain. „A survey on solving cold start problem in recommender systems“. In: *2017 International Conference on Computing, Communication and Automation (ICCCA)*. 2017, pp. 133–138. DOI: 10.1109/CCAA.2017.8229786.

[Had+23] Muhammad Usman Hadi et al. „Large Language Models: A Comprehensive Survey of its Applications, Challenges, Limitations, and Future Prospects“. In: *TechRxiv* 2023.0921 (2023). DOI: 10.36227/techrxiv.23589741.v3. eprint: https://www.techrxiv.org/doi/pdf/10.36227/techrxiv.23589741.v3. URL: https://www.techrxiv.org/doi/abs/10.36227/techrxiv.23589741.v3.

[Hev07] Alan R Hevner. „A three cycle view of design science research“. In: *Scandinavian journal of information systems* 19.2 (2007), p. 4.

[Hou+23] Yupeng Hou et al. „Large language models are zero-shot rankers for recommender systems“. In: *arXiv preprint arXiv:2305.08845* (2023).

[How+20] David M Howcroft et al. „Twenty years of confusion in human evaluation: NLG needs evaluation sheets and standardised definitions“. In: *Proceedings of the 13th international conference on natural language generation*. 2020, pp. 169–182.

[Hua+23] Xu Huang et al. *Recommender AI Agent: Integrating Large Language Models for Interactive Recommendations*. 2023. arXiv: 2308.16505 [cs.IR].

[Hum24] HumanSignal. *Label Studio: Data labeling and annotation tool*. 2024. URL: https://labelstud.io.

[Iiv07] Juhani Iivari. „A paradigmatic analysis of information systems as a design science“. In: *Scandinavian journal of information systems* 19.2 (2007), p. 5.

[JP24] Aryan Jadon and Avinash Patil. „A comprehensive survey of evaluation techniques for recommendation systems“. In: *International Conference on Computation of Artificial Intelligence & Machine Learning*. Springer. 2024, pp. 281–304.

[Jan23] Dietmar Jannach. „Evaluating conversational recommender systems: A landscape of research“. In: *Artificial Intelligence Review* 56.3 (2023), pp. 2365–2400.

[Jan+21] Dietmar Jannach et al. „A survey on conversational recommender systems“. In: *ACM Computing Surveys (CSUR)* 54.5 (2021), pp. 1–36.

[JK02] Kalervo Järvelin and Jaana Kekäläinen. „Cumulated gain-based evaluation of IR techniques“. In: *ACM Transactions on Information Systems (TOIS)* 20.4 (2002), pp. 422–446.

[Ji+23] Ziwei Ji et al. „Survey of hallucination in natural language generation“. In: *ACM computing surveys* 55.12 (2023), pp. 1–38.

[Jia+24] Albert Q Jiang et al. „Mixtral of experts“. In: *arXiv preprint arXiv:2401.04088* (2024).

[KRI07] K KRIPPENDORFF. „Computing Krippendorff's Alpha-Reliability“. In: *http://www. asc. upenn. edu/Krippendorff/* (2007).

[Lei+20] Wenqiang Lei et al. „Conversational Recommendation: Formulation, Methods, and Evaluation“. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '20. Virtual Event, China: Association for Computing Machinery, 2020, pp. 2425–2428. ISBN: 9781450380164. DOI: 10.1145/3397271.3401419. URL: https://doi.org/10.1145/3397271.3401419.

[LI17] International Federation of Library Associations and Institutions (IFLA). *IFLA Statement on Digital Literacy*. Aug. 2017. URL: https://repository.ifla.org/items/f47dc600-7bbb-4ae4-b59e-646cc154ba47.

[Lin+21] Jimmy Lin et al. „Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations“. In: *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*. 2021, pp. 2356–2362.

[Liu22] Jerry Liu. *LlamaIndex*. Nov. 2022. DOI: 10.5281/zenodo.1234. URL: https://github.com/jerryjliu/llama_index.

[Los24] LostRuins. *KoboldCPP: A unified AI writing interface and local LLM host.* https://github.com/lostruins/koboldcpp. 2024.

[Lü+12] Linyuan Lü et al. „Recommender systems“. In: *Physics reports* 519.1 (2012), pp. 1–49.

[Mar25] MariaDB Foundation. *MariaDB Server*. 2025. URL: https://mariadb.com.

[Mat26] Matomo Developers. *Matomo Database Schema Guide*. https://developer.matomo.org/guides/database-schema. Accessed: 2026-04-15. 2026.

[Mye+24] Devon Myers et al. „Foundation and large language models: fundamentals, challenges, opportunities, and social impacts“. In: *Cluster Computing* 27.1 (2024), pp. 1–26.

[oob24] oobabooga. *Text-Generation-WebUI: A Gradio web UI for Large Language Models*. https://github.com/oobabooga/text-generation-webui. Accessed: 2024-04-20. 2024.

[Ope23] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[Ouy+23] Shuyin Ouyang et al. *LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation*. 2023. arXiv: 2308.02828 [cs.SE].

[Pap+02] Kishore Papineni et al. „Bleu: a Method for Automatic Evaluation of Machine Translation“. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Ed. by Pierre Isabelle, Eugene Charniak, and Dekang Lin. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040/.

[PB07] Michael J Pazzani and Daniel Billsus. „Content-based recommendation systems“. In: *The adaptive web: methods and strategies of web personalization*. Springer, 2007, pp. 325–341.

[Pen+23] Baolin Peng et al. *Check Your Facts and Try Again: Improving Large Language Models with External Knowledge and Automated Feedback*. 2023. arXiv: 2302.12813 [cs.CL].

[PCA14] Nicolas Prat, Isabelle Comyn-Wattiau, and Jacky Akoka. „Artifact evaluation in information systems design-science research–a holistic view“. In: (2014).

[Pre26] Preisvergleich Internet Services AG. *Geizhals Österreich*. https://geizhals.at/. Accessed: 2026-04-09. 2026.

[Raf+23] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2023. arXiv: 1910.10683 [cs.LG].

[Rai+24] Mohaimenul Azam Khan Raiaan et al. „A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges“. In: *IEEE Access* 12 (2024), pp. 26839–26874. DOI: 10.1109/ACCESS.2024.3365742.

[RB09] Ehud Reiter and Anja Belz. „An investigation into the validity of some metrics for automatically evaluating natural language generation systems“. In: *Computational Linguistics* 35.4 (2009), pp. 529–558.

[RRS10] Francesco Ricci, Lior Rokach, and Bracha Shapira. „Introduction to recommender systems handbook“. In: *Recommender systems handbook*. Springer, 2010, pp. 1–35.

[Sch+07] J Ben Schafer et al. „Collaborative filtering recommender systems“. In: *The adaptive web: methods and strategies of web personalization*. Springer, 2007, pp. 291–324.

[Al-16] Mohammad Yahya H Al-Shamri. „User profiling approaches for demographic recommender systems“. In: *Knowledge-Based Systems* 100 (2016), pp. 175–187.

[Sha+17] Noam Shazeer et al. „Outrageously large neural networks: The sparsely-gated mixture-of-experts layer“. In: *arXiv preprint arXiv:1701.06538* (2017).

[Sil+19] Thiago Silveira et al. „How good your recommender system is? A survey on evaluations in recommendation“. In: *International Journal of Machine Learning and Cybernetics* 10.5 (2019), pp. 813–831.

[SZ18] Yueming Sun and Yi Zhang. „Conversational Recommender System“. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. SIGIR '18. Ann Arbor, MI, USA: Association for Computing Machinery, 2018, pp. 235–244. ISBN: 9781450356572. DOI: 10.1145/3209978.3210002. URL: https://doi.org/10.1145/3209978.3210002.

[TGL04] Cynthia A Thompson, Mehmet H Goker, and Pat Langley. „A personalized system for conversational recommendations“. In: *Journal of Artificial Intelligence Research* 21 (2004), pp. 393–428.

[TR20] Craig Thomson and Ehud Reiter. „A gold standard methodology for evaluating accuracy in data-to-text systems“. In: *Proceedings of the 13th International Conference on Natural Language Generation*. 2020, pp. 158–168.

[TZ25a] THUDM and Zhipu AI. *GLM-4.5V-106B*. https://huggingface.co/zai-org/GLM-4.5-Air. 2025.

[TZ25b] THUDM and Zhipu AI. *GLM-4.6-355B*. https://huggingface.co/zai-org/GLM-4.6. 2025.

[Tou+23] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].

[Vas+23] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[Wan+23a] Haifeng Wang et al. „Pre-Trained Language Models and Their Applications“. In: *Engineering* 25 (2023), pp. 51–65. ISSN: 2095-8099. DOI: https://doi.org/10.1016/j.eng.2022.04.024. URL: https://www.sciencedirect.com/science/article/pii/S2095809922006324.

[Wan+24] Lei Wang et al. „A survey on large language model based autonomous agents“. In: *Frontiers of Computer Science* 18.6 (2024), p. 186345.

[Wan+23b] Xiaolei Wang et al. „Rethinking the Evaluation for Conversational Recommendation in the Era of Large Language Models“. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 10052–10065. DOI: 10.18653/v1/2023.emnlp-main.621. URL: https://aclanthology.org/2023.emnlp-main.621/.

[TU-26] TU-Wien-dataLAB. *Aqueduct: AI Gateway*. https://github.com/TU-Wien-dataLAB/aqueduct. 2026.

[Zha+25] Yu Zhang et al. „A survey of large language model empowered agents for recommendation and search: Towards next-generation information retrieval“. In: *arXiv preprint arXiv:2503.05659* (2025).

User Study Generation

LLM System Prompt

You are in an E-Commerce Context with the main focus on Hardware. You will be given information about a User-Session on the E-Commerce Platform and you need to transform this information into a singular question that best describes the user intent. Write only the question and not the context. The purpose of this question should be to get one or more products recommended. In the last row you can see what category of product the user wants. If there are one or more filters, include them in your generated question. Please write the question in simple English and in the first-person perspective.

Few-Shot Examples:

- What Intel socket motherboard would you recommend for use with a Samsung SSD?
- Can you recommend me a graphics card that has at least 16 GB of memory?
- What is a good gaming monitor that is compatible with an Nvidia graphics card?
- I have looked at multiple laptops, can you recommend me a good one for gaming?
- Are there any laptops cheaper than 700€ that have an Intel i7 processor?
- Can you recommend me a good gaming mouse that is compatible with a Razer keyboard?
- I have looked at AMD CPUs, can you recommend me a good motherboard for them?
- What is a good gaming headset that is compatible with a PS5?
- Can you recommend me a cheap cable for my iPhone?
- I am looking for a new laptop with at least 16 GB of RAM and a size of 17".
- What is a cheap gaming keyboard that has mechanical switches?

Recommender System Prompts

Table Selection Prompt

Table Selection System Prompt

You are an assistant that selects the most relevant tables for answering a SQL question.

Available tables: [Table Names]

- Identify the product type mentioned in the question.
- Then select the most relevant tables for that product type.
- You can choose multiple tables but keep it to max 5.
- The tables are in German — interpret the user's German question correctly.
- Some tablenames contain specialised terms — use them carefully.
- Return only a comma-separated list of relevant table names.
- Return ONLY the table names; NO explanations.

Question: [User Input]

Schema Refinement Prompt

This prompt is utilised in the second phase of the pipeline to dynamically reduce the context size by selecting only the most relevant columns from the previously identified candidate tables.

Schema Refinement System Prompt

Question: [User Input]

Table Columns: [List of Available Columns]

Return ONLY the relevant column names as a comma-separated list. No other text. No quotes. No spaces. No explanations.

SQL Generation Prompt

SQL Generation System Prompt

Given an input question, create a syntactically correct [Dialect] query.

Use only the following tables (with schema provided): [Table Info]

Rules:

- Think which table and columns are needed to answer the question.
- Select ONLY the columns required to answer the question.
- Do NOT use WHERE conditions if the question does NOT require filtering.
- Use backticks around ALL table and column names.
- Do NOT use table or columns that aren't in the schema.
- ONLY use WHERE conditions if required by the question.
- DO Not use JOIN conditions.
- There are no same type of products in 2 different tables.
- KEEP IT SIMPLE AND VALID.
- Prefer simple queries with minimal conditions.
- DON'T use LIMIT or ORDER BY.
- Always return ONLY the columns named `product` and `product_link`.
- Return ONLY the SQL query; no explanations.

Question: [User Input]

SQL Repair Prompt (Execution Errors)

SQL Repair System Prompt

You are given a user question, schema, a [Dialect] SQL query, and an error message. Return a corrected [Dialect] SQL query.

Rules:

- Use only valid tables/columns from schema.
- Wrap identifiers with backticks.
- Always return ONLY `product` and `product_link`.
- No LIMIT, ORDER BY, ASC/DESC.
- KEEP IT SIMPLE AND VALID.
- Return ONLY the SQL query.
- NO explanations ONLY THE SQL QUERY.

Question: [User Input]
Schema: [Table Info]
Previous SQL: [Generated SQL]
Error: [Execution Error Message]

SQL Retry Prompt (Empty Results)

SQL Retry System Prompt (No Rows)

The last query returned no rows.

Instructions:

- Relax or remove over-restrictive conditions.
- Change the Conditions Where 0 rows are returned.
- Only keep filters strictly required by the question.
- Always return ONLY `product` and `product_link`.
- Do NOT use LIMIT or ORDER BY.
- Use the same tables as before.
- KEEP IT SIMPLE AND VALID.
- Do not use JOIN conditions.
- Use backticks around identifiers.
- Return ONLY the SQL query. No explanations.

Question: [User Input]
Schema: [Table Info]
Previous SQL: [Generated SQL]
Diagnostics: [Feedback/0 Rows Message]

SQL Strict Retry Prompt (Failsafe)

SQL Strict Retry System Prompt

The last query again returned no rows.

Instructions:

- Change/remove the Conditions Where 0 rows are returned.
- Prefer removing filters over changing them.
- Always return ONLY `product` and `product_link`.
- Do NOT use LIMIT or ORDER BY.
- Use the same tables as before.
- KEEP IT SIMPLE AND VALID.
- Do not use JOIN conditions.
- Use backticks around identifiers.
- Return ONLY the SQL query. No explanations.

Question: [User Input]
Schema: [Table Info]
Previous SQL: [Generated SQL]
Diagnostics: [Feedback/0 Rows Message]

Semantic Validation Prompt

Semantic Validation System Prompt

You are a strict MariaDB-SQL validator.

You will be given:

- *A natural language question*
- *A database schema*
- *A candidate SQL query*

Your job:

- Check if the SQL query actually answers the question.
- If numeric/capacity fields exist, they must be used (not just string filters).
- Ensure minimality — no unnecessary tables or filters.
- If correct, return EXACTLY: `VALID`
- Otherwise return a corrected SQL query.
- Check for JOIN conditions — DO NOT use JOINs.
- Check for LIMIT/ORDER BY — DO NOT use them.
- Check that ONLY `product` and `product_link` columns are selected.
- DO NOT WRITE ANY EXPLANATIONS.

Return ONLY:
`VALID`
OR THE SQL QUERY

Question: [User Input]
Schema: [Table Info]
Candidate SQL: [Generated SQL]

Execution-Log-Langchain

LangChain Execution Log

```
Action: sql_db_list_tables
Action Input: ""
Observation: 3m10_19_Schränke_schr19, 19_Barebones_bare19,
19_Gehäuse_geh19, 19_Rack_hd19rack, 19_Rackmount_sys19rack,
3D_Drucker_dr3d, 3D_Druckstifte_dr3dst...
Thought: The question is asking for a 3D printer
recommendation that costs less than 300 (best_price).
To answer this question, I need to query the
"3D_Drucker_dr3d" table in the database.

Action: sql_db_query
Action Input: SELECT * FROM 3D_Drucker_dr3d WHERE
best_price < 300 ORDER BY best_price ASC LIMIT 10
Observation: [('dr3d', '2023-12-12T08:40:14+01:00',
'Creality Ender 3 NEO', ...)]
Thought: The query returned results, but the question
asks for a specific recommendation. I will look at the
results and provide the best option.
Final Answer: The Creality Ender 3 NEO is a recommended
3D printer that costs less than 300.
```

Execution-Log-LLamaIndex

LlamaIndex Execution Trace

```
Trace:  query
  |_templating -> 3e-05 seconds
  |_llm -> 370.2192 seconds
  |_synthesize -> 18.016962 seconds
    |_templating -> 2.6e-05 seconds
    |_llm -> 18.015674 seconds

User Input:  "I want to buy a new CPU from Intel in the
200-300€ range preferable one from the 11th or 12th
generation"

Generated SQL:
SELECT category, product, best_price FROM Intel_cpu1151
WHERE best_price BETWEEN 200 AND 300 AND (category =
'Intel Core i5' OR category = 'Intel Core i7' OR category
= 'Intel Core i9') AND (Architektur = '11th Gen' OR
Architektur = '12th Gen');

Final Answer:  Based on your query, here are some
recommended Intel CPUs within the 200-300€ price range
from the 11th or 12th generation:
1.  Intel Core i5-11400F: This CPU is part of the 11th
generation and is priced around 200€.  It offers 6 cores
and 12 threads, making it suitable for gaming...
```

Execution-Log-LangChain-Agent

LangChain Agent Execution Trace

```
> Entering new AgentExecutor chain...


Action:  sql_db_list_tables
Action Input:
Observation:  10_19_Schränke_schr19, 19_Barebones_bare19,
19_Gehäuse_geh19, 19_Rack_hd19rack, 19_Rackmount_sys19rack,
3D_Drucker_dr3d, 3D_Druckstifte_dr3dst, 3D_Scanner_scan3d,
AMD_Epyc_mbsp3...
Thought:  Question:  What AMD socket motherboard would you
recommend for use with a Samsung SSD 980 PRO?


Action:  sql_db_list_tables
Action Input:
Observation:  10_19_Schränke_schr19, 19_Barebones_bare19,
19_Gehäuse_geh19, ...
Thought:  Question:  What AMD socket motherboard would you
recommend for use with a Samsung SSD 980 PRO?


Action:  sql_db_query_checker
Action Input:  SELECT * FROM `Systemspez_Netzteile_gehsysps`
WHERE `Typ` = 'Mainboards_mit_CPU' AND `Kompatibel` LIKE
'%Samsung%' AND `Kompatibel` LIKE '%980%' AND `Kompatibel`
LIKE '%PRO%' AND `Kompatibel` LIKE '%SSD%' AND `Kompatibel`
LIKE '%AMD%' AND `Kompatibel` LIKE '%Socket%' AND
`Kompatibel` LIKE '%AM4%' AND `Kompatibel` LIKE '%Socket%'
AND `Kompatibel` LIKE '%AM5%' AND `Kompatibel` LIKE
'%Socket%' AND `Kompatibel` LIKE '%PRO%' AND `Kompatibel`
LIKE '%980%' AND `Kompatibel` LIKE '%980%' AND `Kompatibel`
LIKE '%980%' ...
```

Table 1: Comparative retrieval and ranking performance metrics.

Metric	Large (355B)		Medium (106B)		Small (30B)	
	All	Found	All	Found	All	Found
Success Rate	0.670 (492)		0.571 (419)		0.563 (413)	
MRR	0.209	0.311	0.161	0.282	0.191	0.340
P@10	0.377	0.563	0.293	0.513	0.337	0.598
NDCG@10	0.241	0.360	0.186	0.325	0.220	0.391
P@50	0.540	0.805	0.439	0.768	0.461	0.818
NDCG@50	0.278	0.415	0.219	0.383	0.248	0.441
P@100	0.589	0.878	0.484	0.847	0.507	0.901
NDCG@100	0.286	0.427	0.226	0.396	0.256	0.454

Note: "All" denotes systemic performance over the full dataset ($n = 734$), while "Found" isolates the ranking quality of successful retrievals.

Table 2: Operational dynamics and system latency by retrieval outcome.

Model Scale	Outcome	n	Avg. Attempts	Retry Rate	Avg. Latency (s)
Large (355B)	Success	492	1.002	0.20%	9.66
	Failure	242	1.009	0.41%	15.88
Medium (106B)	Success	419	1.007	0.48%	9.79
	Failure	315	1.024	1.27%	22.16
Small (30B)	Success	413	1.007	0.73%	8.94
	Failure	321	1.055	4.67%	12.57

Note: n represents the number of queries; retry metrics and latency illustrate systemic behaviour during successful versus failed attempts.

Table 3: Impact of Temperature (T) and Top-p hyperparameters on retrieval and ranking performance ($n = 734$).

Config		**Hit Rate**		**Systemic Performance (All)**							**Isolated Ranking Quality (Found)**						
T	p	Found	Rate	MRR	P@10	N@10	P@50	N@50	P@100	N@100	MRR	P@10	N@10	P@50	N@50	P@100	N@100
0.25	0.7	488	0.665	0.210	0.376	0.242	0.534	0.279	0.584	0.287	0.316	0.566	0.364	0.803	0.419	0.879	0.431
0.50	0.7	496	0.676	0.211	0.386	0.245	0.549	0.283	0.597	0.291	0.313	0.571	0.363	0.812	0.419	0.883	0.430
0.75	0.7	489	0.666	0.208	0.376	0.240	0.540	0.278	0.579	0.284	0.311	0.564	0.360	0.810	0.417	0.869	0.426
0.25	0.8	491	0.669	0.211	0.381	0.244	0.540	0.281	0.586	0.288	0.316	0.570	0.365	0.807	0.420	0.876	0.431
0.50	0.8	485	0.661	0.209	0.383	0.243	0.542	0.279	0.582	0.286	0.316	0.579	0.368	0.821	0.423	0.880	0.433
0.75	0.8	491	0.669	0.210	0.375	0.241	0.544	0.280	0.591	0.288	0.313	0.560	0.361	0.813	0.418	0.884	0.430
0.25	0.9	493	0.672	0.207	0.376	0.239	0.542	0.277	0.586	0.285	0.308	0.560	0.356	0.807	0.413	0.872	0.424
0.50	0.9	486	0.662	0.199	0.356	0.228	0.530	0.268	0.578	0.276	0.300	0.537	0.345	0.800	0.405	0.872	0.417
0.75	0.9	479	0.653	0.207	0.375	0.240	0.531	0.276	0.572	0.282	0.318	0.574	0.368	0.814	0.423	0.877	0.433

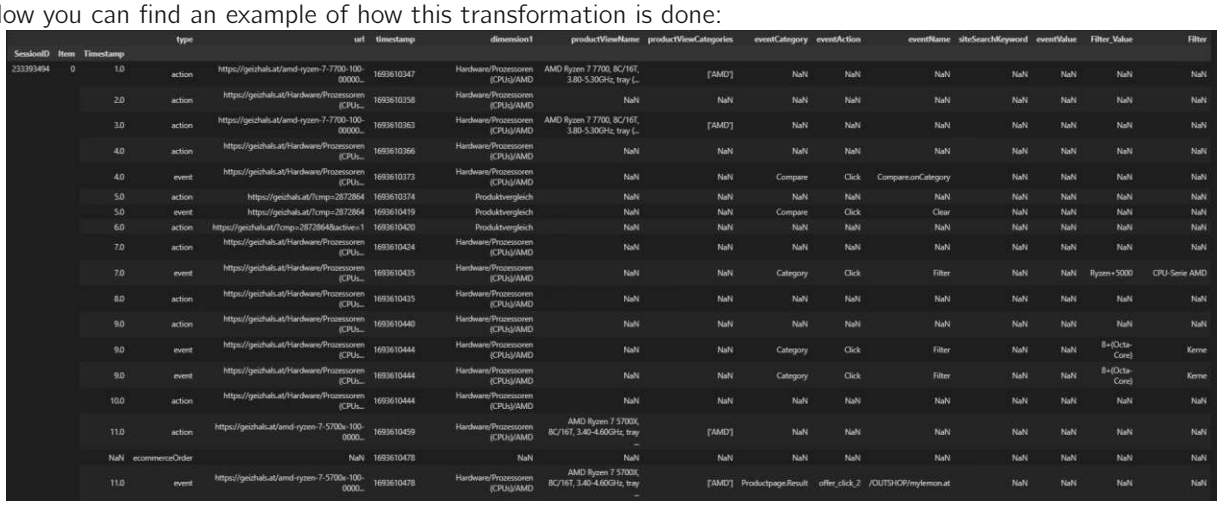
Note: T denotes Temperature, and p denotes Top-p. For spatial formatting, NDCG is abbreviated as "N" in the column headers (e.g., N@10 = NDCG@10). "All" includes failed retrievals as 0, while "Found" isolates successful hits.

User Study For Rec-Sys

Note: You need to label 100 instances for 2.5 bonus points and a further 50 (150 total) an additional 2.5 points (5 total).
We will conduct quality checks of the labeling tasks so please take care of faithfully conducting the study. Irregularities in the labeling can lead to 0 points being awarded.

Context

In this user study, we analyze user behavior data from the price comparison website Geizhals to gain insights into user interactions and preferences. This data is then transformed into textual descriptions to facilitate interpretation by a large language model (LLM). A LLM is then used to generate questions based on the user's browsing activities. These questions are designed to capture the user's intent and preferences for hardware purchases. However, there are many other categories which are available on Geizhals. The resulting dataset of user-generated questions is used to evaluate and enhance a conversational recommender system, focusing specifically on the hardware domain. Below you can find an example of how this transformation is done:



SessionID	Item	Timestamp	type	url	timestamp	dimension1	productViewName	productViewCategories	eventCategory	eventAction	eventName	siteSearchKeyword	eventValue	Filter_Value	Filter
233393494	0	1.0	action	https://geizhals.at/amd-ryzen-7-7700-100-00000...	1693610347	Hardware/Prozessoren (CPUs)/AMD	AMD Ryzen 7 7700, 8C/16T, 3.80-5.30GHz, tray (...	['AMD']	NaN	NaN	NaN	NaN	NaN	NaN	NaN
		2.0	action	https://geizhals.at/Hardware/Prozessoren (CPUs...	1693610358	Hardware/Prozessoren (CPUs)/AMD	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
		3.0	action	https://geizhals.at/amd-ryzen-7-7700-100-00000...	1693610363	Hardware/Prozessoren (CPUs)/AMD	AMD Ryzen 7 7700, 8C/16T, 3.80-5.30GHz, tray (...	['AMD']	NaN	NaN	NaN	NaN	NaN	NaN	NaN
		4.0	action	https://geizhals.at/Hardware/Prozessoren (CPUs...	1693610366	Hardware/Prozessoren (CPUs)/AMD	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
		4.0	event	https://geizhals.at/Hardware/Prozessoren (CPUs...	1693610373	Hardware/Prozessoren (CPUs)/AMD	NaN	NaN	Compare	Click	Compare.onCategory	NaN	NaN	NaN	NaN
		5.0	action	https://geizhals.at/?cmp=2872864	1693610374	Produktvergleich	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
		5.0	event	https://geizhals.at/?cmp=2872864	1693610419	Produktvergleich	NaN	NaN	Compare	Click	Clear	NaN	NaN	NaN	NaN
		6.0	action	https://geizhals.at/?cmp=2872864&active=1	1693610420	Produktvergleich	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
		7.0	action	https://geizhals.at/Hardware/Prozessoren (CPUs...	1693610424	Hardware/Prozessoren (CPUs)/AMD	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
		7.0	event	https://geizhals.at/Hardware/Prozessoren (CPUs...	1693610435	Hardware/Prozessoren (CPUs)/AMD	NaN	NaN	Category	Click	Filter	NaN	NaN	Ryzen+5000	CPU-Serie AMD
		8.0	action	https://geizhals.at/Hardware/Prozessoren (CPUs...	1693610435	Hardware/Prozessoren (CPUs)/AMD	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
		9.0	action	https://geizhals.at/Hardware/Prozessoren (CPUs...	1693610440	Hardware/Prozessoren (CPUs)/AMD	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
		9.0	event	https://geizhals.at/Hardware/Prozessoren (CPUs...	1693610444	Hardware/Prozessoren (CPUs)/AMD	NaN	NaN	Category	Click	Filter	NaN	NaN	8+(Octa-Core)	Kerne
		9.0	event	https://geizhals.at/Hardware/Prozessoren (CPUs...	1693610444	Hardware/Prozessoren (CPUs)/AMD	NaN	NaN	Category	Click	Filter	NaN	NaN	8+(Octa-Core)	Kerne
		10.0	action	https://geizhals.at/Hardware/Prozessoren (CPUs...	1693610444	Hardware/Prozessoren (CPUs)/AMD	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
		11.0	action	https://geizhals.at/amd-ryzen-7-5700x-100-0000...	1693610459	Hardware/Prozessoren (CPUs)/AMD	AMD Ryzen 7 5700X, 8C/16T, 3.40-4.60GHz, tray ...	['AMD']	NaN	NaN	NaN	NaN	NaN	NaN	NaN
		NaN	ecommerceOrder	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
		11.0	event	https://geizhals.at/amd-ryzen-7-5700x-100-0000...	1693610478	Hardware/Prozessoren (CPUs)/AMD	AMD Ryzen 7 5700X, 8C/16T, 3.40-4.60GHz, tray ...	['AMD']	Productpage.Result	offer_click_2	/OUTSHOP/mylemon.at	NaN	NaN	NaN	NaN

This data frame is then transformed into the following text:

The user changed the category to Hardware/Processors(CPUs)/AMD
The user looked at AMD Ryzen 7 7700, 8C/16T,...
The user changed the category to Product-comparison
The user changed the category to Hardware/Processors(CPUs)/AMD
The user changed the filter CPU-Series MAD to Ryzen+5000
The user changed the filter Cores to 8+(Octa-Core)
The user is looking for a product in the category Hardware/Processors(CPUs)/AMD

In the last step, this text is then transformed to the following question using a LLM:

Can you recommend me an AMD Ryzen 5000 processor with at least 8 cores?

CD Lab RecSys

1 Aim of the user Study

The goal of this user study is to verify questions which will be used to evaluate a CRS. The CRS will be evaluated in a way that it has no previous information about the user, his intent or preferences.
Your aim in this user study is to evaluate the created questions based on the given user behavior. When evaluating the generated questions, You should focus on several key aspects to ensure the quality and relevance of the questions:

1. Accuracy

- **Question Representation**: Determine if the question accurately reflects the user intent of the input data (i.e. would a user with the displayed behavior ask this question to a Conversational recommender system (CRS)? Check if any essential information from the input data is missing or incorrectly represented.
- **Content Appropriateness**: Verify that there is no extraneous or irrelevant information in the question that should not be present.

2. Detailing Inaccuracies

If the question is deemed inaccurate, identify specific issues such as:

- **Incorrect Number**: Numerical values in the question that do not match the input data.
- **Incorrect Entity**: Incorrect entities (e.g., product names, categories) mentioned in the question.
- **Missing Number**: Essential numerical information missing from the question.
- **Missing Entity**: Important entities from the input data not included in the question.
- **Hallucination**: Information present in the question that was not part of the input data.
- **Not Checkable**: Aspects of the question that cannot be verified against the input data would take to long to check.
- **Other**: Any other inaccuracies not covered by the above categories.

3. Contextual Relevance

- **CRS Suitability**: Assess whether the question is suitable for use in our specific CRS context. Determine if the question is clear, relevant, and useful for understanding user intent and providing appropriate recommendations.

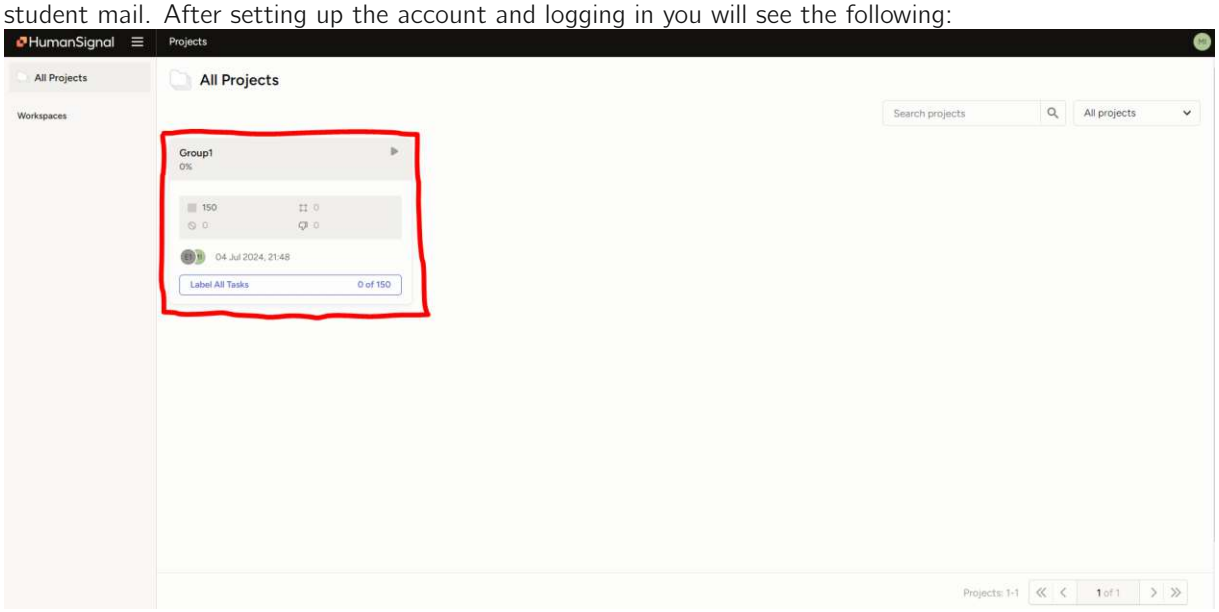
4. Additional Information

- **Additional Comments**: Additionally please feel free to share your thoughts on why you made specific decisions during the study or provide any other feedback or additional information that you think might be helpful. Your comments and insights are greatly appreciated and will help improve the study.

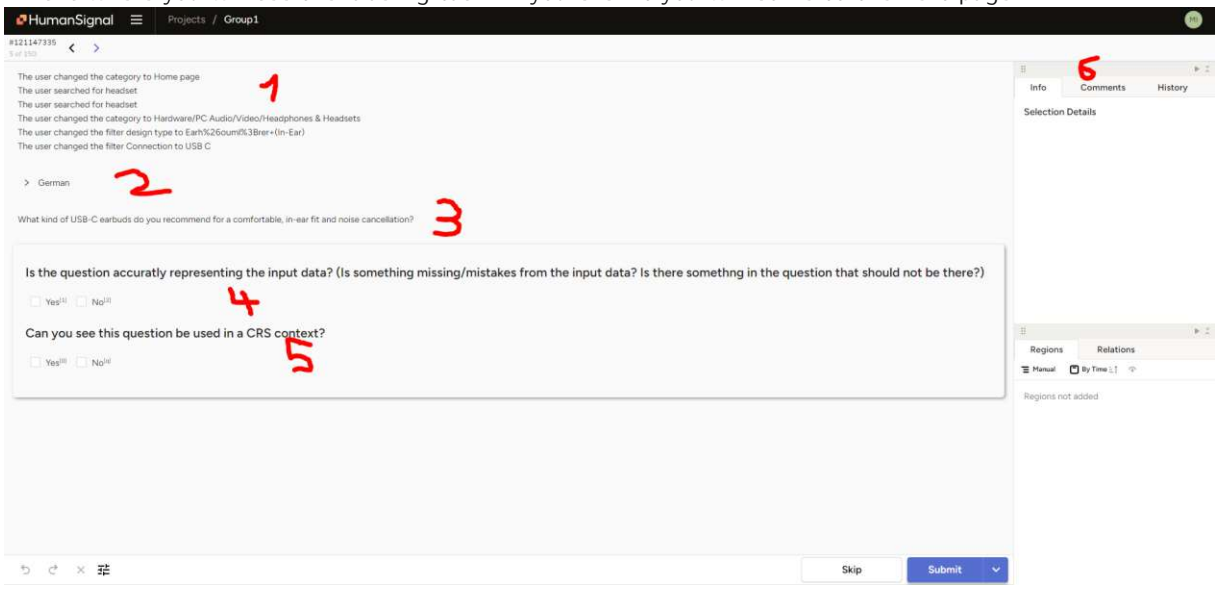
How to

Sign up using the link received via email

You will receive an E-Mail containing an invite link to Label-Studio. Which you then use to sign up using your student mail. After setting up the account and logging in you will see the following:

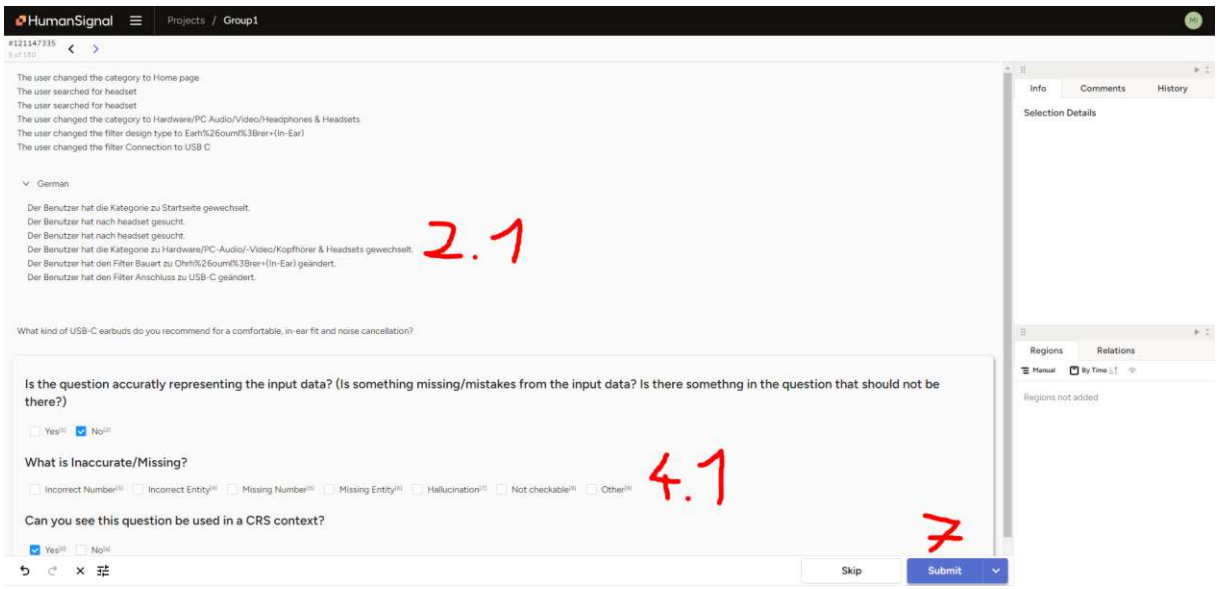


This is where you will see the labeling task. If you click it you will come to the next page:



In the picture above you can see the following:

1. The Data in Text form which the LLM receives
2. A toggleable field that contains the Text in German (How it looks unfolded can be seen in the next picture marked with 2.1)
3. The created question
4. The answer to the Question Representation and Content Appropriateness. In the picture below marked by 4.1, you have the answer fields which ask about the Inaccuracies in detail. This pops up if you answer 4 with a no
5. This is to answer the Contextual Relevance
6. Here you can add your additional information/comments.
7. Finally in the picture below you can press submit after filling out both 4 (additionally 4.1 if no was selected) and 5.



Examples

The user changed the category to Home page The user changed the category to Hardware The user searched for notebook The user changed the category to Hardware/Laptops The user changed the category to Hardware/Laptops/Laptops The user changed the filter Display size up to to up to+14.1%22 The user changed the filter CPU family Intel to Core+i5 The user changed the filter CPU family Intel to Core+i7 The user changed the filter RAM from to from+16GB The user changed the filter Number of M.2 PCIe SSDs to from+1x The user changed the filter RAM type to DDR5
Can you recommend me a good notebook with a Core i7, has at least 16 GB of RAM, DDR5 RAM type, with a display size up to 14.1 inches and has one M.2 PCIe SSD slot?
Is the question accurately representing the input data? (Is something missing/mistakes from the input data? Is there something in the question that should not be there?)

Yes	No
x	o

What is Inaccurate/Missing?						
Incorrect Number	Incorrect Entity	Missing Number	Missing Entity	Hallucination	Not checkable	Other
o	o	o	o	o	o	o

Can you see this question be used in a CRS context?	
Yes	No
x	o

The user changed the category to Home page The user changed the category to Hardware The user searched for notebook The user changed the category to Hardware/Laptops The user changed the category to Hardware/Laptops/Laptops The user changed the filter **CPU family Intel to Core+i5** The user changed the filter RAM from to **from+32GB** The user changed the filter Number of M.2 PCIe SSDs to from+1x The user changed the filter RAM type to DDR5
Can you recommend me a good notebook with a **AMD Ryzen**, has at least **16 GB** of RAM, DDR5 RAM type and has one M.2 PCIe SSD slot?
Is the question accurately representing the input data? (Is something missing/mistakes from the input data? Is there something in the question that should not be there?)

Yes	No
o	X

What is Inaccurate/Missing?

Incorrect Number	Incorrect Entity	Missing Number	Missing Entity	Hallucination	Not checkable	Other
X	X	o	o	o	o	o

Can you see this question be used in a CRS context?

Yes	No
X	o

Here there are two mistakes:

1. It asks for a AMD processor instead of an Intel(Incorrect Entity).
2. The amount of RAM is wrong 16 instead of 32 (Incorrect Number).

The user changed the category to Home page The user changed the category to Hardware The user searched for notebook The user changed the category to Hardware/Laptops The user changed the category to Hardware/Laptops/Laptops The user changed the filter Display size up to to up **to+14.1%22** The user changed the filter CPU family Intel to Core+i5 The user changed the filter **CPU family Intel to Core+i7** The user changed the filter RAM from to from+16GB The user changed the filter Number of M.2 PCIe SSDs to from+1x The user changed the filter RAM type to DDR5
Can you recommend me a good notebook that has at least 16 GB of RAM, DDR5 RAM type, with a specific display size and has one M.2 PCIe SSD slot?
Is the question accuratly representing the input data? (Is something missing/mistakes from the input data? Is there something in the question that should not be there?)

Yes	No
o	X

What is Inaccurate/Missing?

Incorrect Number	Incorrect Entity	Missing Number	Missing Entity	Hallucination	Not checkable	Other
o	o	X	X	o	o	o

Can you see this question be used in a CRS context?

Yes	No
X	o

Here the two mistakes are:

1. The Display size should be 14.1 inches however it only says a specific display size. (Missing Number)
2. The user used a filter to limit the CPU choices to Intel i7s. However this is Missing in the question. (Missing Entity)

The user changed the category to Home page The user changed the category to Hardware The user searched for notebook The user changed the category to Hardware/Laptops The user changed the category to Hardware/Laptops/Laptops The user changed the filter Display size up to to up to+14.1 The user changed the filter CPU family Intel to Core+i5 The user changed the filter CPU family Intel to Core+i7 The user changed the filter RAM from to from+16GB The user changed the filter Number of M.2 PCIe SSDs to from+1x The user changed the filter RAM type to DDR5
Of course I can to that for you: Can you recommend me a good notebook with a Core i7, a **GeForce 4070**, has at least 16 GB of RAM, DDR5 RAM type, with a display size up to 14.1 inches and has one M.2 PCIe SSD slot?
Is the question accuratly representing the input data? (Is something missing/mistakes from the input data? Is there something in the question that should not be there?)

Yes	No
o	X

What is Inaccurate/Missing?						
Incorrect Number	Incorrect Entity	Missing Number	Missing Entity	Hallucination	Not checkable	Other
o	o	o	o	X	o	o

Can you see this question be used in a CRS context?	
Yes	No
X	o

Here both the bold parts are Hallucinations which were not in the original data and were just added by the LLM.(Hallucination)

The user changed the category to Home page The user changed the category to Hardware The user searched for notebook The user changed the category to Hardware/Laptops The user changed the category to Hardware/Laptops/Laptops The user changed the filter Display size up to to up to+14.1 The user changed the filter CPU family Intel to Core+i5 The user changed the filter CPU family Intel to Core+i7 The user changed the filter RAM from to from+16GB The user changed the filter Number of M.2 PCIe SSDs to from+1x The user changed the filter RAM type to DDR5
Can you recommend me a notebook with a Core i7, has at least 16 GB of RAM, DDR5 RAM type, with a display size up to 14.1 inches and has one M.2 PCIe SSD slot and **is similar to my old one**?
Is the question accuratly representing the input data? (Is something missing/mistakes from the input data? Is there something in the question that should not be there?)

Yes	No
o	X

What is Inaccurate/Missing?

Incorrect Number	Incorrect Entity	Missing Number	Missing Entity	Hallucination	Not checkable	Other
o	o	o	o	X	X	o

Can you see this question be used in a CRS context?

Yes	No
X	o

Here the bold part cannot be checked against the data. Additionally You can also say it is a hallucination as nowhere in the data it talks about a previous laptop.

The user changed the category to Home page The user changed the category to Hardware The user searched for notebook The user changed the category to Hardware/Laptops The user changed the category to Hardware/Laptops/Laptops The user changed the filter Display size up to to up to+14.1 The user changed the filter CPU family Intel to Core+i5 The user changed the filter CPU family Intel to Core+i7 The user changed the filter RAM from to from+16GB The user changed the filter Number of M.2 PCIe SSDs to from+1x The user changed the filter RAM type to DDR5
Based on my behaviour what Notebook,Core i7,min 16 GB of DDR5,max 14.1 inches, min 1 M.2 would you recommend?
Is the question accurately representing the input data? (Is something missing/mistakes from the input data? Is there something in the question that should not be there?)

Yes	No
o	X

What is Inaccurate/Missing?

Incorrect Number	Incorrect Entity	Missing Number	Missing Entity	Hallucination	Not checkable	Other
o	o	o	o	X	X	o

Can you see this question be used in a CRS context?

Yes	No
o	X

- Again here is a hallucination that is not checkable.
- Additionally it is not suited for the Use in our specific CRS context as it requires a previous behaviour of the user.

Overview of Tools used

The methodology for utilising these tools the following workflows to ensure original authorship:

- **Software Development Workflow:** The author conceptualised the architecture and logic required to solve specific problems. LLMs were subsequently used to generate localised code snippets, which the author integrated into the broader solution, and to assist in targeted troubleshooting. Additionally, they were used in the documentation process and the writing of inline comments to help keep the codebase understandable and maintainable.
- **Text Composition Workflow:** Initial drafts were written manually by the author and submitted to the AI tools (ChatGPT or Google Gemini) for proofreading and structural reformulation. These suggestions were reviewed and edited by the author. A final pass by the LLMs was used for grammar and spelling verification.

GitHub Copilot
- **Use-Case:** Coding and Software Development.
- **Application:** Assisting with troubleshooting, prototyping of code segments, and generating inline documentation and code comments.

ChatGPT (OpenAI)
- **Use-Case:** Coding, Software Development, and Writing.
- **Application (Coding):** Supporting troubleshooting, code prototyping, and generating documentation.
- **Application (Writing):** Assisting with text composition, specifically by refining sentence structures for clarity, and general proofreading.

Google Gemini
- **Use-Case:** Writing and Data Visualisation (Figure Creation).
- **Application (Writing):** Primarily utilised for proofreading and identifying unclear sections of text for revision and helping with these revisions.
- **Application (Data Visualisation):** Assisting in the translation of evaluation metrics into functional Python scripts to generate the figures presented in the results chapter.