Legal Document Query Language: Conceptualizing Linguistic Commands for AI Assistants in Civil Appeal Proceedings ⋆

Ruben Agazzi[1], Carlo Batini[1], Matteo Palmonari[1] and Monica Vitali[2]

[1]*University of Milano-Bicocca*
[2]*(formerly) Ministery of Justice*

Abstract

In Italy's civil appeal proceedings, judges rely on legal repositories, case management databases, and case filings for decision-making. Generative AI offers promising support for these cognitive tasks but struggles to account for their specificity, limiting effectiveness. In this paper, we present early results on an effort to improve solutions based on generative AI to support second-instance civil proceedings by proposing a Legal Document Query Language (LDQL) for Civil Appeal Proceedings. Inspired by structured query languages, LDQL specifies recurring operations and primitives for such operations. It can be viewed as a conceptual layer to guide the selection of prompts optimized for specific cognitive tasks, the completion of these prompts with textual elements specific to the user request, and the specification of constraints on features that responses should satisfy. As a first contribution, the language helps better clarify the variety of the underlying operations. We discuss a use case where LDQL is employed to interact with specialized prompts with an LLM or an LLM-based system and report about the quality (e.g. accuracy, efficiency, usefulness) of the response in relation to the overall task. Preliminary results suggest that a language like LDQL can support a better orchestration of LLM-based linguistic services thus making it worth proceeding with its implementation using a multi-agent architecture.

Keywords

Legal documents, AI assistants, Query languages, Document management

1. Introduction

In Italy's second-instance civil proceedings, referred here also as appeal civil proceedings, judges consult legal document repositories, case management databases, and key case filings to track procedural events, access precedents, and analyze case materials to decide and motivate it.

Solutions based on generative AI, in short, AI assistants, are promising means to support the cognitive tasks underpinning this consultation and decision-making process, which are left to judges. However, current solutions based on long document processing or even RAG architecture do not explicitly consider the variety of the cognitive tasks performed in the process and hardly capture the specificity of these tasks, which constrain specific linguistic operations on certain documents and data. In this paper, we present early results on an effort to evaluate

SEBD 2025

⋆ You can use this document as the template for preparing your publication. We recommend using the latest version of the ceurart style.

* Corresponding author.

† These authors contributed equally.

and improve AI assistants to support civil appeal proceedings by proposing a Legal Document Query Language (LDQL) for Civil Appeal Proceedings.

Inspired by structured query languages, LDQL specifies recurring operations and primitives for such operations. It can be viewed as a conceptual layer to guide the selection of prompts optimized for specific cognitive tasks, the completion of these prompts with textual elements specific to the user request, and the specification of constraints on the documents to consider or on features that responses should satisfy. As a first contribution, the conceptual language helps better clarify the variety and specificity of the underlying operations. As a second contribution, we use this conceptualization for an early evaluation in terms of reliability and expected utility of AI assistants on an Appeal use case, obtained with the popular ChatGPT assistant and with an on-premise prototype assistant based on a smaller model. Based on the early findings we discuss our plans to extend the prototype exploiting the LDQL conceptualization. The paper is organized as follows. In section2, we discuss the related work, while in Section 3 we introduce a use case in the area of labour that allowed us to identify cognitive activities that can be expressed in terms of verbal commands. Such commands have been a posteriori expressed with a variety of prompts that have been shown to a judge that evaluated them in terms of various qualities. The result of the evaluation is discussed in Section 4, where we discuss the takeaways of the experiment, while in Section 5, we provide a description of the functional architecture of a prototype whose production is in progress. Section 6 briefly addresses future work.

2. Related Work

Languages adopted in document management systems are surveyed in [1]. Speech act theory [2], [3] investigates speech acts; a speech act is created when speaker/writer S makes an utterance U to hearer/reader H in context C. Various classifications of speech acts are provided, and for each item in the classification a list of acts is provided. A paper affine to ours is [4], where a set of commands, some of them similar to ours, are defined to compare question answering on a set of documents spanning from traditional search engines to LLMs. Through a survey, they find that users prefer search engines over LLMs for high-stakes queries, where concerns regarding information provenance outweigh the perceived utility of LLM responses. Furthermore, they define a set of qualities associated to commands similar to ours.

Think aloud techniques are widely adopted to elicit various aspects of human activity, including in the legal domain. In [5], lawyers' interactive information behaviour is discussed; in [6], specialist legal expertise is measured through a think-aloud verbal protocol analysis.

Various surveys appeared recently in the literature on Prompt engineering, e.g., a huge amount of techniques are discussed in [7]. Literature on specific commands is rich, especially for the Search command and for the Summarize command, e.g., in [8], where the two classical Extractive Abstractive techniques are compared on different aspects. In [9], a command resulting from the composition of our Summarize and Integration commands is applied to build a unique text from several overlapping and contradictory documents.

AI-powered assistants such as Copilot are now appearing, providing some of our commands such as Summarize, integrated in platforms such as in the case of Copilot in Microsoft 365.

3. Experimentation phase: eliciting the commands with a think aloud approach and conceiving and experimenting the commands

To elicit linguistic commands that are most relevant in expressing cognitive activities of a judge in a civil procedure we focused on one case study in the area of labour and one judge, limiting our investigation to documents shown in Figure 1. For reach of the First degree Judgement, Second degree Appeal and Second degree Brief we consider the text of the document, in which we assume are mentioned several precedent judgements related to similar cases from Courts, Courts of Appeal and Court of Cassation, see Figure 1.

By means of interviews, we were able to reconstruct all the actions (Figure 2) related to the drafting of the judgment, expressed in terms of text fragments corresponding to Statement of facts, Precedent citations, legal reasoning (ground) and rulings. Such text fragments can be inherited from documents in the case file or citations of judgements in court, court of appeal, and court of Cassation, or else, as to the ruling, the result of the autonomous development of the judge's reasoning.

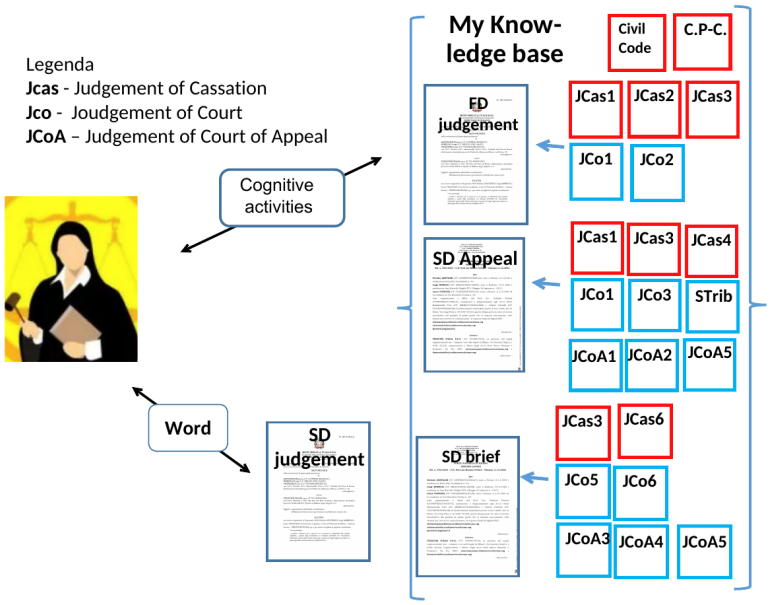

Figure 1: Cognitive activities related to most relevant documents

Adopting a think-aloud approach, we associated one or more cognitive activities with each action, which we have expressed in terms of linguistic commands, see the list of the eleven commands in the left-hand part of Figure 3.

At this point, we asked ChatGPT Scholar to produce a list of Acts of Speech for document management in the legal domain; the result was a list that included e.g. Search and Summarize, but also procedural acts such as Register (a judgment). Using a technique based on in-context prompting, with a further prompt we asked to produce a list with, e.g., create and without, e.g.,

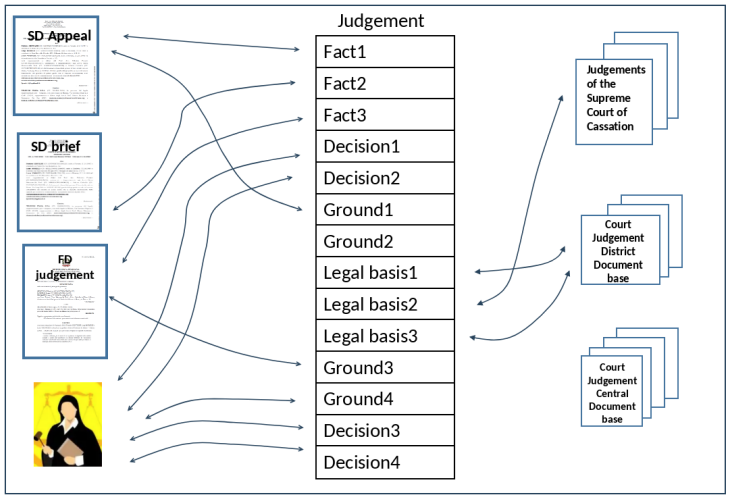

Figure 2: Fragments of the parts of the judgement and their provenance from a. documents in the case file, b. judgements or c. judge autonomous reasonong.

register, resulting in responses to the fourteen acts of speech in the right-hand side of Figure 3.

We also mapped the two lists of commands and acts. It may be seen that the two acts of Speech Argue and Deduce have no counterpart in our list; this is reasonable, since we excluded, coherently with the rules of the European AI Act, actions in which Generative AI can play an active and autonomous role in sentencing. The Order command is expressed in LDQL with the lexicographic condition, There are three linguistic commands, namely, Calculate, Expand and Integrate, that have no counterpart in the Acts of Speech; Calculate computes simple mathematical expressions, Expand is the complementary command w.r.t. Summarize, Integrate results in the fusion of several documents or part of them into a unique text.

To proceed with the experimentation, we have first produced a syntax for the eleven commands. Following the syntax of the Select instruction in relational databases: `SELECT <what (among columns)>, FROM <where (among relations)>, WHERE <condition>`, we defined a similar general syntax for commands of LDQL shown in the left-hand part of Figure 4. In relational databases SELECT refers to a unique command, which is replaced in LDQL by the set of different commands shown in Figure 3. Notice that the *What* part (the output of the command) spans over a significant set of categories, Entity and Concept are the typical categories of the targets of semantic enrichment and search; other categories cover the most relevant documents and semantic parts of documents resulting from the experimentation on the case considered. The *Where* part refers to all categories in the *What* part, except Entity and Concept, including the three documents of the Appeal case file we mentioned in Figure 1. Finally, the *How* (condition) part in this first specification of LDQL corresponds to logical, mathematical, temporal, spatial and organizational operators; a formal definition of the syntax is out of scope in this paper.

Figure 4 shows the syntax of atomic commands (Search, Summarize, etc.). Besides atomic commands, LDQL supports composite commands, which have a functional nested syntax, in which the Where part of a command C1 is expressed in terms of a second command C2, e.g.,

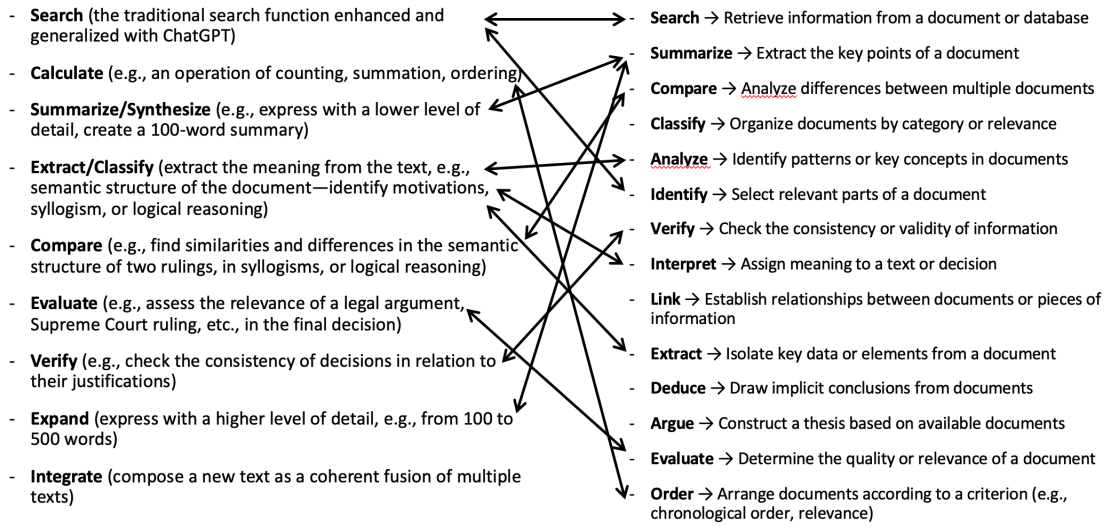

Figure 3: Linguistic commands identified and correspondences with acts in speech act theory applied to legal document

Command	Where	What	How (Condition)
Search Summarize Extract Compare Calculate Explore Connect Expand Evaluate Verify Integrate	- First Degree Judgment - Appeal - Brief Draft Judgment - Article - Civil Code - Code of Civil Procedure - Decisions - Excerpt - Facts - Reasoning / Grounds - Logical Step - Logical Path - Precedent - Supreme Court Precedent - Lower Court Precedent - Premise - Party's Requests / Party's Claims - Reference - Syllogism	Entity Concept Number Article •Civil Code •Code of Civil Procedure Decisions Excerpt Facts Reasoning / Grounds Logical Step Logical Path Precedent Supreme Court Precedent Lower Court Precedent Premise Party's Requests / Party's Claims Reference Syllogism	- Lexicographic . Logical - Mathematical - Organizational - Temporal - Spatial

Figure 4: General structure of commands

Summarize(Search "..."). A formal representation of simple and composite commands in terms of a syntax, e.g. in Backus Normal Form, could easily be provided, but it is out of the scope of this paper.

To proceed with the experimentation, we have focused on the first four commands, Search, Summarize, Extract, and Compare, and for each of them we produced from six to ten prompts. Prompts for the Search and Summarize commands are shown in Tables 2 and 3, where we will discuss their evaluation. Notice that prompts 1, 2, and 6 of the Summarize command are composite commands, and in one of them we used the command Calculate.

A significant aspect of LDQL is the set of qualities that each prompt should respect. We distinguish two types of properties, respectively "command-driven" properties and "user-driven properties", see Table 1.

Table 1
Evaluation measures associated with different commands

Quality/Command	Who evaluates	Search	Summarize	Extract/Classify	Compare
Accuracy	Developer	x			
Completeness	Developer	x			
Effectiveness	User		x	x	x
Efficiency	User	x	x	x	x
Utility	User	x	x	x	x

Command driven properties are accuracy, namely the precision of the information provided in output (e.g. an excerpt is faithful to the original) and completeness, meaning that all texts requested have been produced in output. Command-driven properties are defined for the Search command and have to be evaluated by testing the prompt with different examples of where and what. User driven properties are evaluated by the user. Efficiency and Utility are common to all commands; efficiency is the perceived saved time when using the command, while utility is the perceived added value for the user in his/her activity. Effectiveness is defined for Summarize, Extract and Compare, and represents the perceived adherence of the output to user needs; e.g., for Summarize, the summary must contain all the relevant aspects of the original document, and hide the details.

In the following, we report preliminary results of system and user evaluations. Tables 1 and 3 show the evaluations for the search (nine prompts) and summarize (seven prompts) commands, referring to their qualities as indicated in Table 1. Accuracy and completeness are evaluated in scale 0-1, while remaining qualities are evaluated with a scale where 1 corresponds to "not at all", 2 corresponds to "a little", 3 corresponds to "somewhat", and 4 corresponds to "a lot".

Table 2: Search command prompts and their evaluation; NM = not measured (yet)

Search command prompts	**Accuracy**	**Completeness**	**Efficiency**	**Utility**
Search for and list the legitimacy precedents in Judgment S1.	NM	1	3	2
Search for the Supreme Court precedents cited in Judgment S2, listed in chronological order.	0.8	0.6	3	3
Search in S2 for Supreme Court rulings from 2018 onward.	0.75	NM	2	3
Search in S2 for the text quoted between " " from the Supreme Court judgment.	NM	1	4	4
Search in S1, R2, M2, and S2 for the citation of Judgment Cass. October 17, 2018, No. 26017.	NM	NM	4	4
Search for the precedents cited in the appeal (or memorandum) that are not present in the first-instance civil judgment.	1	NM	4	4
Search for all the merit precedents in the appeal, extract the quoted sentences, and report them in quotation marks.	1	1	3	3
Search in the appeal for merit precedents issued by the Civil Courts of Northern Italy.	1	NM	3	3
Search for the civil code articles cited in the first-instance judgment and retrieve the texts of such articles.	0.75	NM	2	2
Average	**0.88**	**0.90**	**3.11**	**3.11**

Table 3: Summarize command prompts and their evaluation

Summarize command prompts	**Effectiveness**	**Efficiency**	**Utility**
Search in S1 and summarize the text quoted between " " from the Supreme Court judgment Cass. October 17, 2018, No. 26017.	3	4	4
Calculate the number of words in the citation (50) and summarize the citation text in fewer than 30 words.	3	3	3
Summarize the rulings of Judgment S1 in fewer than 50 words.	4	4	4
Summarize very long sentences associated with quoted judgments in S1.	4	4	4
.Calculate the number of words in the phrases between " " related to the judgments cited in S1, search for the judgment number containing the phrase "contested, in the case," and summarize in fewer than 100 words the sentence in S2 related to the judgment.	2	2	3
Search for all the following precedents of merit in the Appeal, extract the quoted sentences, reporting them in quotation marks, and for each, summarize the legal reasoning.	4	3	4
Average	**3.33**	**3.33**	**3.67**

In Table 4, we show the aggregated results for the entire set of four commands in terms of the user-evaluated criteria. Contrary to the initial intuition, the command Compare is considered less efficient and useful w.r.t. other commands, and contributes significantly to lower the global evaluations. Interestingly, the Effectiveness quality has the highest average score, meaning that ChatGPT in the perception of the user, “made a very good job”, independently from time saved and value for the user of the commands. A more comprehensive analysis appears in the next section.

Table 4
Evaluation of different tasks based on various criteria

	Effectiveness	Efficiency	Utility	Average
Search	not defined	3.1	3.1	3.1
Summarize	3.3	3.3	3.7	3.4
Extract	3.8	3.2	3.4	3.5
Compare	3.6	2.5	1.9	2.7
Average	3.6	3.0	3.0	

4. Takeaways of the experimentation phase

Beyond the assessments provided, the experimentation phase was experienced by the judge-user as a process of learning, acquiring knowledge, and practicing a new language for interacting with the available knowledge within civil proceedings. This experience has led to the following reflections.

The ultimate purpose of applying Generative AI to civil proceedings is to extract knowledge from the procedural documents submitted by the parties and the evidence produced during the trial, thereby assisting the judge in formulating the final decision.

In line with this objective, the Search command—whose utility and efficiency are evident—demonstrates enhanced effectiveness over traditional keyword-based search due to its ability to incorporate temporal conditions (e.g., retrieving the most recent decisions of the Court of Cassation or the Court of Justice of the European Union from a specified year onward) as well

as spatial/organizational conditions (e.g., filtering decisions issued by Courts of Appeal within one or more regions). The combination of these two types of conditions further enhances its efficacy.

Regarding the Summarize command, its usefulness and efficiency are significantly leveraged following the entry into force of Ministerial Decree No. 110 of August 7, 2023. This decree, issued by the Ministry of Justice, establishes criteria for drafting, setting word limits, and structuring judicial documents by defining the fields necessary for entering information into the procedural database. The decree imposes word limits on party submissions (such as petitions and pleadings) and similarly guides judges in drafting their judgments. Consequently, in the process of selecting documents for inclusion in judgments, it has become extremely beneficial to use functionalities that enable the extraction of summarized portions of procedural documents within a specified word limit. Even more useful is a functionality that by using the Summarize and Expand commands together, continuously generates versions with increasing or decreasing word limits allowing the judge to select the most appropriate word count.

Both the Extractive and Abstractive types of the command Summarize have proven to be useful, depending on the specific needs they serve. For example: Extractive summarization is particularly beneficial during the case study phase, while Abstractive summarization is more appropriate during the drafting of the judgment itself— which is inherently dialogical. Also, Extractive summarization may be preferable when dealing with novel cases, whereas Abstractive summarization may be better suited for cases where prior rulings exist.

In this regard, combining the Summarize command with Search to target specific sets of documents, as described above, further enhances its utility.

Since the Extract command is designed to retrieve aspects of a document relevant to its meaning by analyzing words and phrases linked by syntactic rules, it is particularly useful for extracting the facts of the case from party submissions and the first-instance judgment, as well as identifying the syllogism or logical reasoning proposed by the parties for the judge's consideration.

When used in conjunction with the Compare command, the Extract command proves to be both efficient and useful for addressing specific judicial needs. In particular, it aids in:

- Assessing the necessity of an evidentiary investigation: If the factual accounts extracted from the parties' submissions align, the need for fact-finding at the first-instance stage is obviated.
- Second-instance proceedings: If no discrepancies exist between the facts stated in the first-instance judgment and those presented in the appeal (including syllogisms or the logical reasoning of the parties), the Compare command can help pinpoint the exact point of divergence and extract the decisive logical aspect of the opposing arguments.

The preliminary conclusion of this experimentation phase is that, beyond the ability to use linguistic commands individually, these commands should be regarded as a toolbox that can be utilized in various sequences depending on the scenario and the specific objectives of the judge.

This approach should consider:

- The quantity, quality, and diversity of the knowledge to be processed;
- The homogeneity or divergence of the legal materials;

- The maturity, emerging nature, stability, instability, or obsolescence of the legal framework governing rulings both at the preliminary and at the adjudicative stages.

5. Prototype

In this experiment, a human has used a top-tier LLM with exceptionally long context via a chatbot, deciding which document(s) should be passed to the LLM for each prompt. Documents were anonymized beforehand.

Our goal is to develop a system that exploits the conceptualization provided by LDQL can handle different prompts with dedicated prompt processing. We want a system that can operate with an LLM hosted on-premises, avoiding sharing sensitive data with third-party companies and having reasonable efficiency constraints (an LLM as small as possible). We are developing such a system as an evolution of DAVE, a prototype web application that combines semantic search and a conversational interface, which can exploit an on-premise small LLM and a Retrieval Augmented Generation (RAG) framework.

DAVE is designed to analyze document collections in knowledge-intensive domains, mixing features that cover both directions in the extractive-abstractive spectrum discussed in [4]. It provides a graphical user interface (GUI) that facilitates search, visualization, exploration and interactions with the documents in the knowledge base. The system adopts an entity-centric approach, by annotating the input documents with entities found using entity extraction pipelines. Describing these algorithms in detail is out of the scope of this paper; the reader can refer to our previous work where different versions of these pipelines are defined in detail [10, 11, 12]. As a result of this pre-processing step, documents are annotated with named entities of different types (e.g., Person, Organization, Location, Date, Money, etc.). On top of these annotated documents DAVE offers the following key functionalities.

1. **Entity-Driven Faceted Search**: The search process begins with a textual keyword search, retrieving an initial set of relevant documents. Users can then refine their queries by leveraging named entities associated with the documents in the annotations, allowing for more precise filtering and structured retrieval.
2. **Conversational Interface**: DAVE includes a chatbot, i.e., a conversational interface that enables users to interact with the document collection through natural language prompts. The underlying system, initially developed to provide question-answering functionalities, is based on the Retrieval-Augmented Generation (RAG) framework. The retrieval module incorporates named entities to enhance the retrieval of relevant chunks, which are subsequently used for answer generation. The user can select which retrieval strategy is used to answer the question (e.g., with or w/o entities), including a "no retrieval" option.
3. **Document Viewer**: DAVE includes a built-in document viewer that allows users to read the full text of a document along with its corresponding entity annotations. Users can inspect named entity mentions, modify entity links to the Wikipedia knowledge base, and view or edit clusters of mentions that refer to the same entity, enabling more refined entity management and knowledge integration. Therefore users can correct mistakes made by the entity extraction process and consolidate the knowledge base interactively.

4. **Integration of Functionalities**: The system enables seamless integration between faceted search and the conversational interface. For instance, users can first identify a set of relevant documents using the entity-driven faceted search and then use the chatbot on top of the selected set of documents; in this case, only chunks from the selected documents are considered for generating the system's responses (filters are applied by the RAG's retrieval module). This helps the user narrow the focus of the chatbot interactively. A special case of this integration is when the chatbot is activated from the Document Viewer: in this case, the chatbot only looks at chunks within the document that the user is exploring.

To use DAVE on the Appeal use case, we added a new feature: the chatbot can bypass the RAG module, and consider the whole document(s) as a context to respond to the user's prompt. Obviously, the possibility to use this feature is subject to whether the documents fit into the size of the context window of the underlying LLM; if the document exceeds the context limit, the RAG module is activated.

5.1. Early experiments on the Appeal use case and considerations on LDQL

In this section, we report on preliminary experiments conducted to use DAVE in the Appeal use case. The objective of these experiments is to obtain preliminary insights into the capabilities that a small on-premise LLM exhibits on the same conditions in which we tested ChatGPT. The response generation in our current prototype is performed using the *Phi-3.5-mini-ITA* model, a fine-tuned Italian-language version of Microsoft's *Phi-3.5-mini* [13], with a 128k-token context window. The model is deployed on a server equipped with a single NVIDIA Tesla T4 GPU, featuring 16 GB of VRAM. We employed a quantized version of *Phi-3.5-mini-ITA* to optimize VRAM usage. While we are aware of novel and promising LLMs that work with Italian, we selected this model because we empirically found that it achieves a good trade-off between quality of responses, speed and size, especially when quantized.

Preprocessing. Initially, all documents were processed using the document annotation pipeline to extract named entities (NER), link entities to Wikipedia where possible (NEL), identify entities without a corresponding Wikipedia entry (NIL Prediction), and cluster entity mentions referring to the same entity [10]. After processing, the documents were indexed in an Elasticsearch database, segmented into chunks with a maximum length of 500 tokens, and assigned embeddings computed using the *gte-multilingual-base* embedding model from *Alibaba-NLP* [14].

DAVE usage. After processing and indexing all documents, we tested DAVE's conversational interface by submitting the same set of questions previously posed to ChatGPT, with exactly the same prompts, and collecting the responses. To use a setting as similar as possible to the one used int he experiments with ChatGPT4o, we used the new functionality we introduced, where DAVE tries to use the full document as context if it fits the context window, and the RAG otherwise. We identified the relevant documents with search and constrained the chatbot to work with the selected documents. We employed Retrieval-Augmented Generation (RAG) for only 5 out of the 16 prompts, specifically those related to the appeal document, as it is the longest.

Focusing on the evaluation that can be performed by system developers (see Table 1, we found that most of the responses were reasonably accurate and complete, though their quality was generally lower compared to ChatGPT's GPT-4o model, particularly for the summarization and extraction commands. For certain tasks, such as the count command, the model did not produce satisfactory results. This limitation is likely due to the inherent architecture of the model: *Phi-3.5-mini* is significantly smaller than ChatGPT's GPT-4o, with only 3.82 billion parameters compared to the estimated 1.76 trillion parameters of GPT-4o. Additionally, we used a quantized model due to hardware memory constraints.

A challenge we encountered was the limited context length available due to hardware memory constraints. Although *Phi-3.5-mini* supports a 128k-token context window, we were only able to utilize a maximum of 20,000 tokens. This is a significant disadvantage compared to ChatGPT, which has the computational resources to process the full text of all documents within its context for answer generation. Despite this limitation, we were able to fit almost all documents within the available context. For the longest document (the appeal document), where the full text exceeded the context limit, we used the RAG framework, where we retrieve the most relevant chunks using a hybrid retrieval method that combines vector search, full-text search, and entity-based search. While this constraint posed a challenge, the model still handled these cases quite effectively.

5.2. What's next? LDQL and multi-agent RAG

The preliminary experiments suggest that ChatGPT and DAVE return promising responses when fed with whole documents as context, but this naif prompting method is not reliable enough for some tasks (e.g., it does not retrive all precedents in S2 - see Table 2) and can hardly scale to more complex cases with more and longer documents to consider. Observe that in the experiments, the user has selected the documents to consider for generating the response. However, we would like a system that determines where information should be searched. Also, no specific strategy other than some heuristics has been used to optimize the prompts. While this specific use case has documents of moderate length (yet, one exceeded the memory constraints in our infrastructure), appeal decisions may need to work with longer documents. This means that some sort of RAG framework needs to be considered. However, previous work has found that pushing the user response to the retrieval module as-is is frequently ineffective [15, 16]. And in fact, several approaches are proposed to process the user prompts and documents with more complex pipelines, e.g., [17, 18]. The LDQL provides a good conceptualization to handle these challenges, by conceptualizing the user needs in a structured format. This format suggest that extracting specific elements from user prompt could be beneficial: different commands may be associated with different prompts; the What part, may be used by the retrieval module; the Where part, may be translated in filters on documents to consider for the reply.

We plan to address all these observations by moving from a naive RAG architecture to a modular RAG architecture backed by a multi-agent system [19], which is depicted, together with two screenshots, in Figure 5. In this architecture, users' prompt are interpreted by one agent that, using the LDQL as a reference, extracts specific elements and generate and route queries (e.g., to the RAG module) optimized for specific commands. Other agents may need to verify and cross check the responses, another pattern applied by RAG systems in other

domains [17, 18, 19].

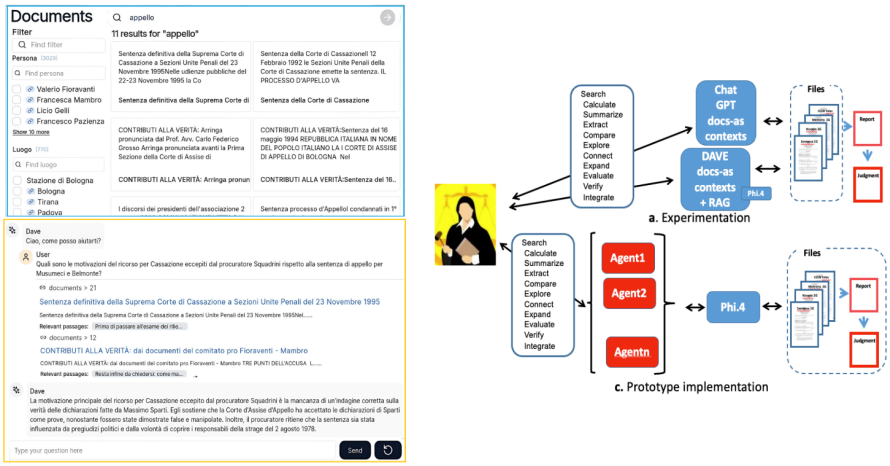

Figure 5: Two examples of screenshots of DAVE (left part) - Early experiments (this paper) to an LDQL-based assistant (right part)

6. Conclusions

In this work, we have presented a project for defining a verbal command language that extends the structure of the Select statement for databases to documents. We extend the command into a set of eleven verbal commands that express the cognitive needs of a judge in a civil appeal trial. We present the outcomes of an experiment, in which a judge has evaluated prompts based on five of the eleven commands in terms of effectiveness, efficiency and utility. We finally describe the architecture of a prototype whose production is in progress. Next phases of the project will be the completion of the experimentation with the remaining six commands and the full production of the prototype.

Preliminary results suggest that a language like LDQL can support a better orchestration of LLM-based linguistic services thus making it worth proceeding with its implementation using a multi-agent architecture. While the current LDQL proposal refers to Civil Appeal Proceedings, we believe that similar complex cognitive tasks may be useful in other domains both internal and external to Justice.

Acknowledgments

This work is part of a research project conducted with the CINI consortium and funded by the Italian Ministry of Justice. It is also partially funded by the European innovation actions enRichMyData (HE 101070284) and DataPACT (HE 101189771), and the Italian project Discount Quality for Responsible Data Science (PRIN 202248FWFS), funded by the European Community - Next Generation EU.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] C.-O. Truică, E.-S. Apostol, J. Darmont, T. B. Pedersen, The forgotten document-oriented database management systems: An overview and benchmark of native xml dodbmses in comparison with json dodbmses, Big Data Research 25 (2021) 100205.

[2] J. R. Searle, Expression and meaning: Studies in the theory of speech acts, Cambridge University Press, 1979.

[3] K. Allan, P. Lamarque, R. Asher, Speech act theory: Overview, Concise encyclopedia of philosophy of language (1997) 454–66.

[4] T. Worledge, T. Hashimoto, C. Guestrin, The extractive-abstractive spectrum: Uncovering verifiability trade-offs in llm generations, arXiv preprint arXiv:2411.17375 (2024).

[5] S. Makri, A. Blandford, A. L. Cox, This is what i'm doing and why: Methodological reflections on a naturalistic think-aloud study of interactive information behaviour, Information Processing & Management 47 (2011) 336–348.

[6] P. J. Macmillan, Thinking like an Expert Lawyer: Measuring Specialist Legal Expertise through Think-Aloud Problem Solving and Verbal Protocol Analysis, Ph.D. thesis, Bond University, 2015.

[7] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, A. Chadha, A systematic survey of prompt engineering in large language models: Techniques and applications, arXiv preprint arXiv:2402.07927 (2024).

[8] G. Sharma, D. Sharma, Automatic text summarization methods: A comprehensive review, SN Computer Science 4 (2022) 33.

[9] G. Lior, A. Caciularu, A. Cattan, S. Levy, O. Shapira, G. Stanovsky, Seam: A stochastic benchmark for multi-document tasks, arXiv preprint arXiv:2406.16086 (2024).

[10] R. Pozzi, R. Rubini, C. Bernasconi, M. Palmonari, Named entity recognition and linking for entity extraction from italian civil judgements, in: R. Basili, D. Lembo, C. Limongelli, A. Orlandini (Eds.), AIxIA 2023 - Advances in Artificial Intelligence - XXIInd International Conference of the Italian Association for Artificial Intelligence, AIxIA 2023, Rome, Italy, November 6-9, 2023, Proceedings, volume 14318 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 187–201. URL: https://doi.org/10.1007/978-3-031-47546-7_13. doi:10.1007/978-3-031-47546-7_13.

[11] V. Bellandi, C. Bernasconi, F. Lodi, M. Palmonari, R. Pozzi, M. Ripamonti, S. Siccardi, An entity-centric approach to manage court judgments based on natural language processing, Computer Law Security Review 52 (2024) 105904. doi:10.1016/j.clsr.2023.105904.

[12] R. Pozzi, V. Barbera, R. Alva Principe, D. Giardini, R. Rubini, M. Palmonari, Combining knowledge graphs and nlp to analyze instant messaging data in criminal investigations, in: M. Barhamgi, H. Wang, X. Wang (Eds.), Web Information Systems Engineering – WISE 2024, Springer Nature Singapore, Singapore, 2025, pp. 427–442.

[13] M. Abdin, J. Aneja, H. Awadalla, A. Awadallah, A. A. Awan, N. Bach, A. Bahree, A. Bakhtiari,

J. Bao, H. Behl, et al., Phi-3 technical report: A highly capable language model locally on your phone, arXiv preprint arXiv:2404.14219 (2024).

[14] X. Zhang, Y. Zhang, D. Long, W. Xie, Z. Dai, J. Tang, H. Lin, B. Yang, P. Xie, F. Huang, et al., mgte: Generalized long-context text representation and reranking models for multilingual text retrieval, in: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track, 2024, pp. 1393–1412.

[15] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, H. Wang, H. Wang, Retrieval-augmented generation for large language models: A survey, arXiv preprint arXiv:2312.10997 2 (2023).

[16] Y. Zhou, Y. Liu, X. Li, J. Jin, H. Qian, Z. Liu, C. Li, Z. Dou, T.-Y. Ho, P. S. Yu, Trustworthiness in retrieval-augmented generation systems: A survey, arXiv preprint arXiv:2409.10102 (2024).

[17] S. Semnani, V. Yao, H. Zhang, M. Lam, WikiChat: Stopping the hallucination of large language model chatbots by few-shot grounding on Wikipedia, in: H. Bouamor, J. Pino, K. Bali (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2023, Association for Computational Linguistics, Singapore, 2023, pp. 2387–2413. URL: https://aclanthology.org/2023.findings-emnlp.157.

[18] Z. Xu, M. J. Cruz, M. Guevara, T. Wang, M. Deshpande, X. Wang, Z. Li, Retrieval-augmented generation with knowledge graphs for customer service question answering, in: Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2024, pp. 2905–2909.

[19] A. Singh, A. Ehtesham, S. Kumar, T. T. Khoei, Agentic retrieval-augmented generation: A survey on agentic rag, arXiv preprint arXiv:2501.09136 (2025).