Interactive AI for Computer Algebra: An Agentic Assistant for `ore_algebra` *

Lixin Du
Institute for Algebra, Johannes Kepler University, Linz, Austria, A4040
lixindumath@gmail.com

Abstract

We present a natural language assistant for the `ore_algebra` package in SageMath. Given a user query, the system retrieves relevant documentation, generates executable SageMath code with a Large Language Model (LLM), executes it, and returns the result together with source-level citations. The assistant combines a local documentation index with an agentic Retrieval Augmented Generation (RAG) pipeline and offers three execution modes (Auto, Fast, and Plan) to balance latency and multi-step reasoning. The current prototype supports common `ore_algebra` tasks, including GCRD and LCLM, closure properties of D-finite functions, guessing from data, and Gröbner basis computations. Exposed through an interactive web interface, the system is intended to reduce the need for manual lookup of function usage in specialized symbolic computations.

1 Introduction

Ore algebras provide a unified framework for linear differential and recurrence operators and are central to the study of D-finite functions [10, 6, 3]. Packages for working with Ore algebra include, among others, `OreTools` [1], `DEtools`, `LREtools`, and `gfun` [9] for Maple, the `HolonomicFunctions` package by Koutschan [7] for Mathematica, and the `ore_algebra` package [4, 5] for SageMath (Sage) [11].

While these systems provide powerful symbolic capabilities, they present a steep learning curve, typically requiring users to select appropriate methods and specify correct syntax. Recent advances in Large Language Models (LLMs) [2] allow users to pose mathematical queries in natural language, but general-purpose models do not reliably handle the detailed syntax and method inventory of specialized libraries such as `ore_algebra`, often selecting inappropriate or unsupported methods.

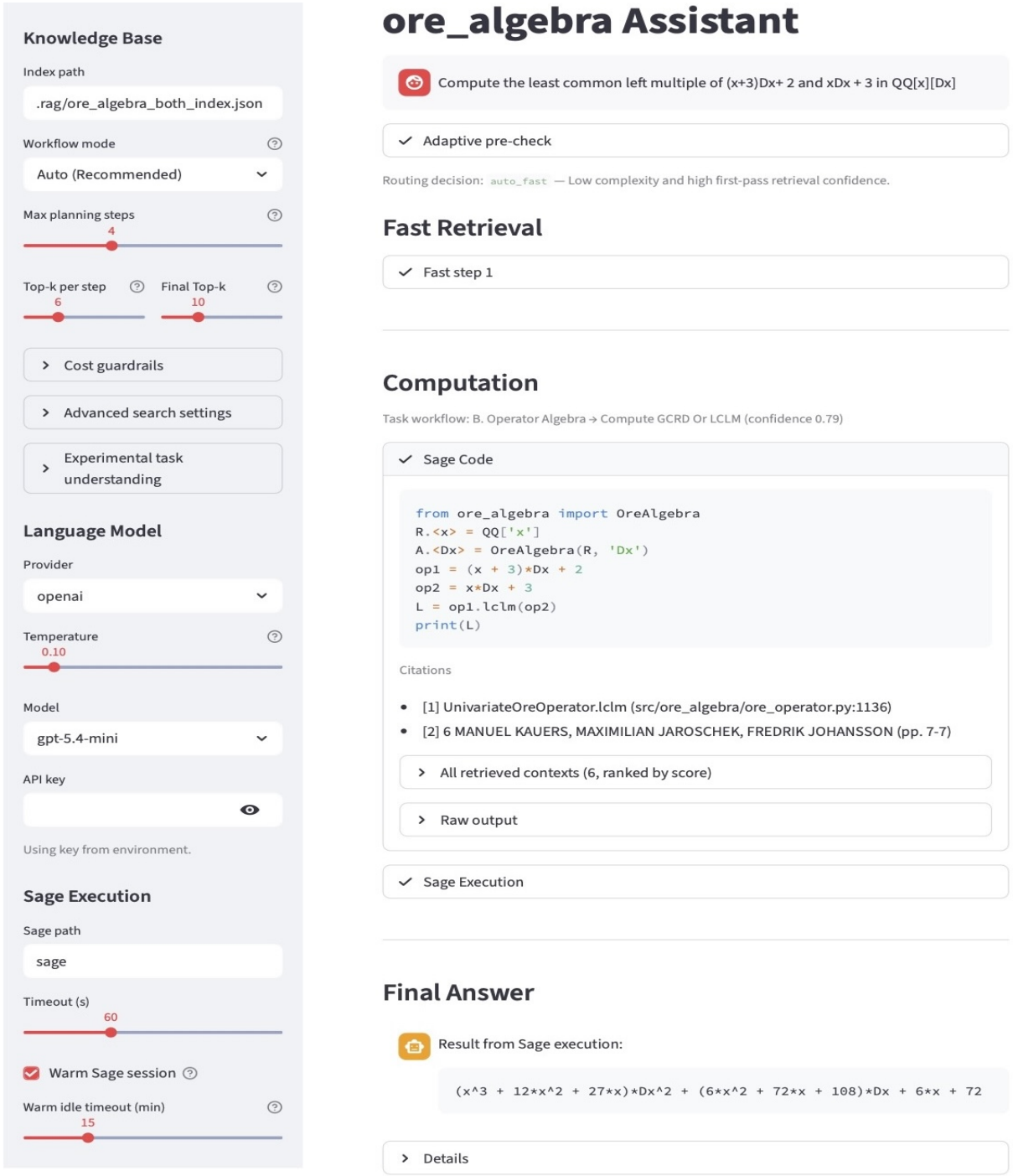
To bridge this gap, we apply Retrieval Augmented Generation (RAG) [8] to develop an interactive assistant for the `ore_algebra` package. Our system searches a knowledge base built from generated documentation, user guides, source-code docstrings, and examples. It uses the retrieved context to generate executable `ore_algebra` code, runs it, and returns the computation result together with citations for the methods used. The current prototype covers common

*L. Du was supported by the Austrian FWF grant 10.55776/PAT9952223.

task families in `ore_algebra`, including GCRD and LCLM, closure properties of D-finite functions, guessing from data, and Gröbner basis computations. The software can be downloaded from: `https://github.com/LixinDu/ore-algebra-agentic-rag-assistant`

2 An Example Screenshot

The screenshot below shows that for a natural-language request, the assistant generates and executes SageMath code and returns source citations.



3 System Overview

The system translates natural-language queries into executable computations through the following pipeline:

- **Knowledge Base:** We index the user-facing guides of `ore_algebra`, together with docstrings, mathematical examples, and cross-references extracted from the package source code.
- **Retrieval:** An incoming query is searched against this knowledge base using a hybrid approach: keyword matching for exact method names and semantic search over dense sentence embeddings for conceptual mapping.
- **Code Generation:** The top retrieved documentation snippets are passed to an LLM together with the user's original query. The model is instructed to produce self-contained SageMath code based only on the provided context.
- **Execution:** After syntax validation, the code runs in a SageMath environment with a fixed timeout. The mathematical output, the executed code, and the corresponding documentation citations are then surfaced to the user.

The implementation is accompanied by a hand-curated workflow benchmark spanning the workflow families currently supported by the prototype.

4 Workflow-Guided and Planning-Based Reasoning

The central difficulty in generating correct code is method selection: identifying which `ore_algebra` methods match a user's query. To handle requests of varying complexity, the system offers three modes:

- **Fast (Workflow-Guided) Mode:** maps common queries directly to predefined workflows and usually requires only one or two retrieval steps.
- **Plan (Planning-Based) Mode:** decomposes harder requests into subtasks and iteratively retrieves supporting documentation.
- **Auto Mode:** estimates query complexity and chooses between Fast and Plan.

These modes are complementary: workflow-guided reasoning provides rapid execution for standard queries, while planning-based reasoning acts as a fallback for more complex queries.

5 Discussion and Future Work

Beyond being a computational tool, this project also serves as a case study in LLM-assisted software engineering for computer algebra. The prototype was developed with substantial use of LLM-based programming assistants, including OpenAI Codex and Anthropic Claude. In our experience, these tools significantly accelerated the prototyping of the web interface and retrieval pipeline, while the mathematically delicate components still required careful human guidance and benchmark-driven testing.

The current limitations of the prototype—including occasional failures in method selection and multi-step computations—thus reflect not only open engineering challenges, but also present limitations of LLM-assisted development in this setting. At the same time, this system already lowers the barrier to using `ore_algebra` by translating natural-language requests into executable SageMath code.

Future work includes extending coverage to tasks such as annihilating ideals and identity proving, together with more systematic evaluation across workflows. In addition, we are exploring a "D-finiteness tutor" that can answer theoretical questions and invoke the current system as a backend solver for concrete computations.

Acknowledgement. The author would like to thank Manuel Kauers for valuable suggestions.

References

[1] S. A. Abramov, H. Q. Le, and Z. Li. OreTools: a computer algebra library for univariate Ore polynomial rings. Technical Report CS-2003-12, University of Waterloo, 2003.

[2] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–51, 2024.

[3] M. Kauers. *D-Finite Functions*. Springer, 2023.

[4] M. Kauers, M. Jaroschek, and F. Johansson. Ore polynomials in Sage. In *Computer Algebra and Polynomials*, pages 105–125. Springer, 2015.

[5] M. Kauers and M. Mezzarobba. Multivariate Ore polynomials in SageMath. *ACM Communications in Computer Algebra*, 53(2):57–60, 2019.

[6] M. Kauers and P. Paule. *The Concrete Tetrahedron*. Springer, 2011.

[7] C. Koutschan. HolonomicFunctions (user's guide). Technical Report 10-01, RISC Report Series, Johannes Kepler University, Linz, Austria, 2010.

[8] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

[9] B. Salvy and P. Zimmermann. Gfun: a Maple package for the manipulation of generating and holonomic functions in one variable. *ACM Transactions on Mathematical Software*, 20(2):163–177, 1994.

[10] R. P. Stanley. Differentiably finite power series. *European journal of combinatorics*, 1(2):175–188, 1980.

[11] The Sage Developers. *SageMath, the Sage Mathematics Software System*, 2024. `https://www.sagemath.org`.