

mathCrow: Agentic AI for Math Tasks

EE-526 Final Project Report

Ahmed Ali Abbasi

Electrical and Computer Engineering, Iowa State University

Ames, IA, USA

aabbasi1@iastate.edu

Abstract—We propose mathCrow, a proof-of-concept extension of ChemCrow [2] to math-based tasks. mathCrow is an agentic framework with access to two expertly designed tools and the ability to independently determine which tool to call to accomplish a given task. This framework provides greater autonomy than a human-based framework, where a human expert would parse the input prompt, decide which tool (function or software package) to call and preprocess the data according to the requirements of the selected tool. We provide an open-source/zero cost implementation of mathCrow using Meta’s llama3.3 model and Huggingface embeddings (in addition to a simpler implementation with paid OpenAI LLM models and embeddings). We test our proposed mathCrow framework on text datasets derived from two research papers and with complex linear assignment problems (LAP). Our results show that mathCrow i) reduces hallucinations in LLM generated responses, and ii) increases autonomous task completion by formulating a precise linear assignment problem from a complex input prompt. Interestingly, our results also confirm that, for the reasoning-based LAP task, GPT-o reasoning models perform better than their general counterparts.

Index Terms—Agentic AI, ChemCrow, MathCrow, LangGraph

I. INTRODUCTION

Large Language Models (LLMs) are becoming increasingly common. Today, these models are being widely used across the entire spectrum of modern human society. Ranging from personal use for everyday tasks to large-scale adoption by banks, tech-giants, and industry for complex and sophisticated applications, the use of LLMs is ever expanding. Indeed, this is as profound a revolution as those induced by the internet or the personal computer. As such, the goal of augmenting and improving the abilities of LLMs has received a lot of attention from the research community. Out of the several ideas to improve an LLM’s abilities, which we discuss in detail below, one is agentic AI. In this project report, we propose mathCrow and investigate the use of agentic AI for math-based tasks.

A. Related work

Next, we discuss the broad themes of research aimed at improving the capabilities of LLMs. Chain of Thoughts (CoT) [5] and Tree of Thoughts (ToT) [6] are two prompt-engineering techniques to improve the planning/reasoning ability of LLMs. These approaches aim to design a prompt that encourages the

Acknowledging HPC Nova Cluster for providing free access to A-100 GPU. Acknowledging [1] for example code of Agentic RAG.

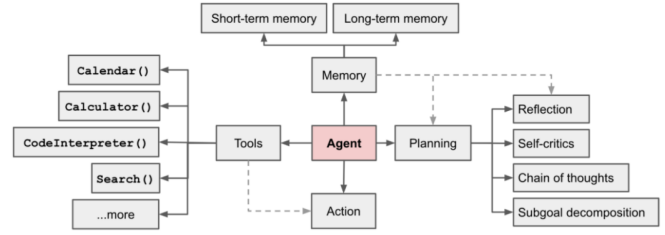


Fig. 1: Overview of an LLM powered autonomous agent system. An agent (LLM) is equipped with 3 components: memory, planning and tools. These components are intended to enhance the agent’s (or LLM’s) task-solving abilities. Given a task, the agent decides how to use these components to solve the task (or take action). Since the agent (or LLM) decides how to use its components, the LLM is also referred to as the ‘brain’. In this work, we propose and implement the mathCrow agent with Tools and Memory components. Figure from [3].

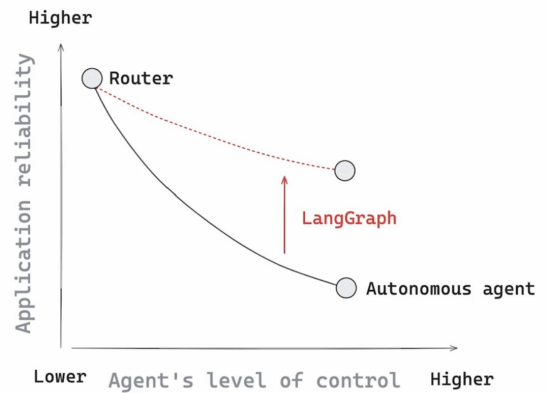


Fig. 2: Routers have minimum autonomy and are implemented by “if-else” conditions, which ensures higher reliability. Agents delegate all decision making and planning to the brain (LLM) and provide greater autonomy. Langgraph is a python package used to implement agentic frameworks. Langgraph allows interpolating between the two extremes by merging human feedback with agentic decision making. Figure from [4].

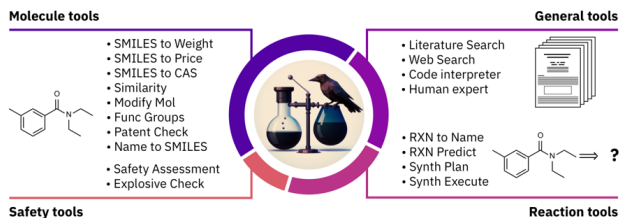


Fig. 3: ChemCrow [2] designs an agent specialized to chemistry tasks. Figure shows example of 18 tools categorized into 4 domains {Molecule, General, Safety, Reaction}. Figure from [2].

LLM to think step-wise. In the case of [5], a single prompt is input to the LLM by the user, and the LLM thinks (or does the steps) internally, but does not output its thoughts. For [6], the steps or thoughts are output by the LLM and analyzed before being used to generate subsequent prompts in a multi-prompt approach.

Retrieval Augmented Generation (RAG) [7] addresses two challenges with stand-alone LLM use: i) small context window and ii) the need for fine-tuning to update the LLM’s information or to specialize it for a particular application/dataset. RAG overcomes these challenges by using large external long-term-memory storage to store domain/application specific information. This database is then searched for the relevant information depending on the input prompt/ query.

Agentic AI improves the autonomous decision-making ability of AI frameworks, Fig. 1. Essentially, an agent is informed of the components at its disposal, for example: these could be RAG for information retrieval, CoT and ToT for Planning, and other specialized tools for domain/application specific tasks. Given a task, the agent, which is an LLM, forms its own plan to complete the task using the components at its disposal, Fig. 2. ChemCrow [2] is an agent that is integrated with 18 expertly designed tools for chemistry specific tasks Fig. 3. This is the first work to establish the superiority of a specific framework to the general GPT-4 framework for academic/research task Fig. 4. It is also the most relevant work to our project, the proposed mathCrow agentic framework.

In the rest of this section, we introduce mathCrow and discuss its utility and application. In section II, we provide a detailed description of mathCrow and the components and tools it uses, followed by a high level implementation in section III, results in section IV, and concluding remarks in section V.

B. Our contribution: mathCrow

1) *Description of mathCrow:* mathCrow is a proof-of-concept extension of the recently proposed ChemCrow agentic framework to math-based tasks. We integrate two components with mathCrow: the two tools: i) RAG and ii) LAP solver tool and one long-term memory component: FAISS database [8]. We describe both tools and the Faiss database in more detail in Section II.

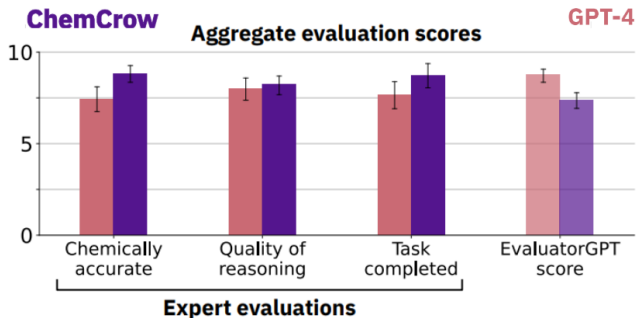


Fig. 4: ChemCrow, with expertly designed tools, is compared to GPT-4 on 3 metrics: chemical accuracy, quality of reasoning, and task completion. Human experts rank the output of ChemCrow higher. Interestingly, a separate evaluator LLM ranks GPT-4’s output higher. Note that the mean evaluator GPT score is plotted and that the evaluator score of each of the three metrics is not provided separately. Figure from [2].

2) *Novelty compared to ChemCrow:* There are two challenges that needed to be overcome for the mathCrow agentic framework. First, mathematical texts include a large number of LaTeX typset equations. The performance of retrieval augmented generation (RAG) was unknown for such corpora of text. Second, math-based tasks require more reasoning and may be more challenging for the LLM parse to parse/understand. Indeed, as we demonstrate in Tool Results 2 in section IV, the standard GPT model fails for the more complicated linear assignment problem whereas the reasoning gpt-4o model does not.

3) *Utility of mathCrow:* For each of the two tools, agentic RAG and LAP solver, we justify their utility over standard I/O ChatGPT prompting. The RAG tool overcomes two problems with standard LLM use: i) context window restriction and ii) hallucination. The external database used to store the documents against which the input query is compared (the context) may comprise hundreds of gigabytes of data, equal to millions of tokens, well beyond the maximum 128,000 tokens context window of the latest LLMs. By evaluating whether or not the retrieved context is indeed relevant to the input query, and answering only if the context is relevant, RAG also reduces hallucinations. However, this hallucination reduction technique is also possible with standard GPT prompting, provided that the context is small enough. Lastly, if the LLM is deployed locally, then sensitive documents can also be analyzed using RAG, without compromising privacy.

The LAP tool has two key benefits. 1) It can solve high-dimensional linear assignment problems (of size 10,000 x 10,000 or more) because the computation is done on local hardware, not on openAI servers. 2) The agent is able to formulate a precise linear assignment problem from a word statement. This eliminates the need to parse and understand the problem statement, define a cost matrix as a numpy array, and then pass it as an argument to the function. See Tool Results 2 in section IV for an example where it is not straightforward to define the cost matrix.

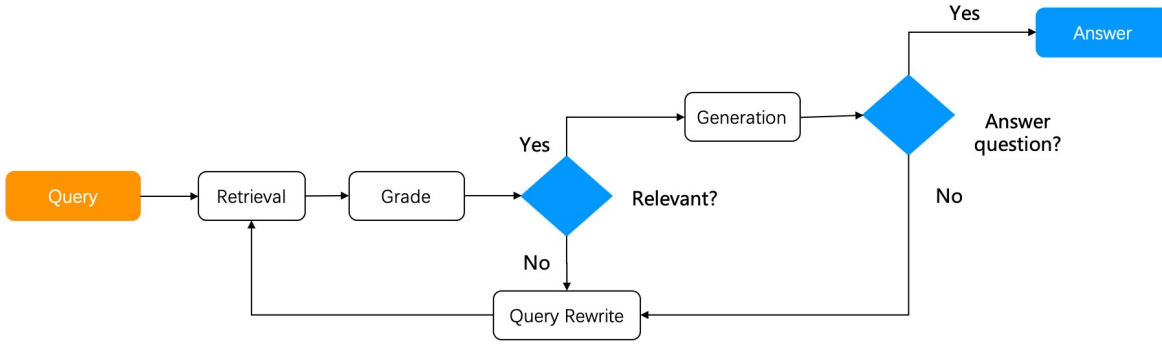


Fig. 5: Retrieval Augmented Generation (RAG) with a Grader. RAG uses an LLM to grade if the retrieved data is relevant to the query. This grading step is key to reducing hallucination. For a description of how we integrate RAG with our agent, see the discussion in Sec II and Fig. 6. Figure from [7].

We also note that, given an input task, the agent fully determines the appropriate tool to call, with no human intervention or feedback. This shows the inherent autonomy of agentic AI frameworks. Finally, our easy-to-use agentic framework for mathematics, mathCrow, lowers the barriers for non-experts to use math-specific complex software and makes it easier for experts and not, alike, to prune long research papers for a quick and accurate understanding.

II. MATHCROW DESCRIPTION

The MathCrow agent has access to two components, tools and memory. We describe each of these components below.

A. Component 1: Tools Description

1) *Agentic-Retrieval Augmented Generation (RAG)*: First, we describe the difference between agentic RAG and RAG. A thorough comparison between the two can also be found at [9]. While both methods involve retrieving context from an external vector database Fig. 5, agentic RAG, where recall that the agent is essentially an LLM, rewrites the query before searching for relevant context in the vector database 5. This is crucial because user/input queries to the vector database are often in the form of questions whereas the content of the database is not. To rewrite the query, the agent uses the description of the retrieval tool, which is a brief description of the data/documents stored in the vector database.

2) *LAP solver*: The LAP solver tool is implemented by wrapping the ‘linear_sum_assignment’ function from the ‘scipy.optimize’ library as a tool. Wrapping a function as a tool, which is possible using Langgraph, makes these functions callable for the agent (or LLM). Langgraph also allows a description of the of the function arguments so that the agent can design these arguments appropriately when calling it.

B. Component 2: Memory/Vectorstore Description

The vector store is an external storage that contains the dense embeddings of the text. A key design requirement of the vector store is that it should allow a fast similarity search of the stored text with the input query. There are two widely used vector dataaabaes: FAISS from Meta [8] and

Chroma [10]. It is important to note that the input query and the documents in the database are compared based off of the numerical embeddings. This is unlike other word-based frequency matching methods such as TF-IDF.

C. Algorithm

We implement mathCrow with Langgraph, which implements agentic frameworks using states, nodes and edges. States are represented by a global variable, usually a list of strings. The graph begins at the ‘start’ node with the state variable equal to input query. The state is then passed from the start node, through a sequence of nodes, to the ‘end’ node. The state flow (transitioning between nodes) is controlled by edges. Nodes and edges are both represented by python function. Typically, nodes modify the state (by appending the list of messages) and edges analyze, given the current state, the next node to transition to. For a visual representation of the mathCrow agentic framework, see Fig. 6.

III. MATHCROW IMPLEMENTATION

We implement our framework in Python. The two main niche packages we use are Huggingace and Langgraph. The necessary licenses required are API-keys from Huggingface (free), OpenAI (paid), and Ollama (free).

A. Using Open-AI Paid API and Google-Colab (Paid Tier)

We evaluated the performance of our agent using three models: ‘gpt-4.1’, ‘gpt-4.1-mini’, and ‘gpt4-o’. While the inference/chat completion code runs on the openAI servers, the vectorstore embeddings are still computed on the local hardware. This is why we had to use a GPU supported Google Colab run-time.

B. Using Ollama and ISU Nova HPC Cluster

For the open-soure (zero-cost) local implementation, we swap OpenAI models with open-source meta llama3.3, which is available at Ollama [11]. An important consideration here is that we have to use quantized models with a size of 49 GB that can fit onto the NOVA A-100 GPU, see Fig. 7.

IV. RESULTS

A. Datasets and preprocessing

We selected the two papers [12] and [13] to test mathCrow on. These papers were selected to avoid querying knowledge the LLM might already contain. This ensured that the output answers were indeed based off of the retrieved context. PDFs cannot be split into documents, and were converted to markdown or text using Llama Parse [14]. The text documents were chunked with RecursiveCharacterTextSplitter from Langchain [4]. Embeddings were generated using the thenlper/gte-small model, available for free at [15].

B. Hyperparameters

We provide details on the hyperparameters in Table I. All hyperparameters apply to the RAG tool. The LAP solver tool does not have any hyperparameters to tune.

We comment on the three sets of results individually below.

1) *Results for Tool 1: RAG1-AltGDMin [12]*: 4 of the 5 answers provided are correct. It is remarkable that mathCrow is able to also answer the finer questions regarding minute details such as difference between altGDMin and two other closely-related algorithms, convex optimization and alternating minimization. The answer to question 4, however, is incorrect. The mathematical expression returned is for the expected value of the gradient, $(X - X^*)B^\top$, not the objective function $\|(X - X^*)_\Omega\|_F$. It is possible that this incorrect answer may be a consequence of the inherent difficulty of having to parse mathematical text. Note that the answer to question 5 shows that the agent-generated response does not hallucinate.

2) *Results for Tool 2: RAG2-Strawberry [13]*: The answers are correct, and also brief. Again, we see that the RAG tool is able to find minute information such as benchmark algorithms and quantification metrics.

3) *Results for Tool 3: LAP Solver*: The first linear assignment problem (LAP) is easier to parse and the agent (gpt 4.1) is able to correctly define the cost matrix so that the final solution is also correct. The second LAP is more difficult. GPT 4.1 does not define the correct matrix. Particularly, the constraint ‘drone D2 must be assigned to either Zone Z2 or Z3’ is not incorporated. It is encouraging to see that the reasoning model gpt-4o defines the correct cost matrix and returns the correct assignment.

V. CONCLUSION AND FUTURE WORK

1) *Project Success*: Learnt Langgraph to deploy agentic frameworks, gained hands-on experience with OpenAI API, local LLM deployment with Ollama, Huggingface, and HPC. To the best of our knowledge, this project is the first implementation of RAG for mathematical texts.

2) *Strengths and Weaknesses*: Agentic RAG minimizes hallucinations. The LAP solver provides greater flexibility and autonomy as strings (words and sentences) can be passed without specifying cost matrices. The weaknesses are that mathCrow has access to a limited number of tools, requires hyperparameter tuning (chunk size and chunk overlap), and is sensitive to agent (LLM) capabilities.

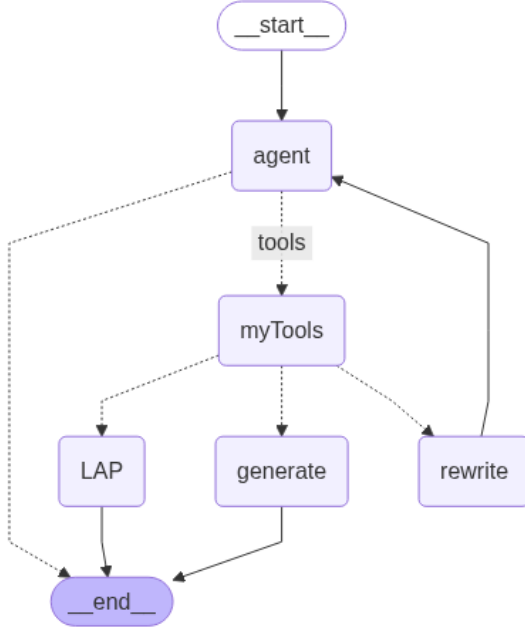


Fig. 6: Figure shows nodes and edges. ‘__start__’ node receives user/input prompt (solid edge) and passes the input prompt to node ‘agent’. ‘agent’ analyzes the prompt to decide which/if any tool to call (dotted conditional edge). If the LAP tool was called, the flow routes to LAP node, which displays the output of the LAP. If the RAG tool was called, the retrieved context is evaluated to be either relevant or not. If the context is relevant, the final response is generated at the ‘generate’ node. Otherwise, the flow transitions to the ‘rewrite’ node and the ‘agent’ is called again.

llama3.3:70b-instruct-q5_0	522f1b464c62	49GB	128K context window	Text input	5 months ago
llama3.3:70b-instruct-q5_1	118a0cbe95b2	53GB	128K context window	Text input	5 months ago
llama3.3:70b-instruct-q5_K_M	a495e09a0513	50GB	128K context window	Text input	5 months ago
llama3.3:70b-instruct-q6_K	992c05c90cd8	58GB	128K context window	Text input	5 months ago
llama3.3:70b-instruct-q8_0	d5b5e1b84868	75GB	128K context window	Text input	5 months ago
llama3.3:70b-instruct-fp16	8fbdb61705a0	141GB	128K context window	Text input	5 months ago

Fig. 7: Notice the reduced model size (49 GB) for the 5-bit quantized version compared to 141 GB for the floating point 16 version. The difference is crucial as only the smaller (49 GB) model fits on a single Nvidia A-100 GPU that has 80 GB of VRAM.

Hyperparameter	Description	Value
Chunk size	Size (in tokens) of each chunk or segment stored in the database.	200
Chunk overlap	Overlap between consecutive chunks.	20.
Embedding model	Model that embeds tokens in the chunks to vectors.	thenlper/gte-small
Vectorstore	External long-term memory storage {Faiss or Chroma}.	FAISS
Top- k	Number of document chunks retrieved from the vectorstore as context.	5

TABLE I: Table of hyperparameters along with their descriptions and values.

3) *Lessons learned*: Always use the class project to learn something new.

4) *Future Work*: Currently, mathCrow has access to the agentic retrieval augmented generation (RAG) tool and the Linear Assignment Problem (LAP) solver tool and we leave integration with a higher number of tools as future work.

REFERENCES

- [1] Agentic RAG — langchain-ai.github.io. https://langchain-ai.github.io/langgraph/tutorials/rag/langgraph_agentic_rag/. [Accessed 15-05-2025].
- [2] Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6(5):525–535, 2024.
- [3] Lilian Weng. LLM Powered Autonomous Agents — lilianweng.github.io. <https://lilianweng.github.io/posts/2023-06-23-agent/>. [Accessed 23-04-2025].
- [4] Why LangGraph? — langchain-ai.github.io. https://langchain-ai.github.io/langgraphjs/concepts/high_level/. [Accessed 14-05-2025].
- [5] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [6] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.
- [7] <https://github.com/yinfeng>. Agentic RAG - Definition and Low-code Implementation — RAGFlow — ragflow.io. <https://ragflow.io/blog/agentic-rag-definition-and-low-code-implementation>. [Accessed 14-05-2025].
- [8] GitHub - facebookresearch/faiss: A library for efficient similarity search and clustering of dense vectors. — github.com. <https://github.com/facebookresearch/faiss>. [Accessed 14-05-2025].
- [9] Agentic RAG: turbocharge your RAG with query reformulation and self-query! - Hugging Face Open-Source AI Cookbook — huggingface.co. https://huggingface.co/learn/cookbook/en/agent_rag. [Accessed 14-05-2025].
- [10] GitHub - chroma-core/chroma: the AI-native open-source embedding database — github.com. <https://github.com/chroma-core/chroma>. [Accessed 14-05-2025].
- [11] llama3.3 — ollama.com. <https://ollama.com/library/llama3.3>. [Accessed 13-05-2025].
- [12] Namrata Vaswani. Altgdm: Alternating gd and minimization for partly-decoupled (federated) optimization, 2025.
- [13] Ruolin Liu and Julie Dickerson. Strawberry: Fast and accurate genome-guided transcript reconstruction and quantification from rna-seq. *PLoS computational biology*, 13(11):e1005851, 2017.
- [14] LlamaParse: Transform unstructured data into LLM optimized formats — LlamaIndex - Build Knowledge Assistants over your Enterprise Data — llamaindex.ai. <https://www.llamaindex.ai/llamaparse>. [Accessed 13-05-2025].
- [15] MTEB Leaderboard - a Hugging Face Space by mteb — huggingface.co. <https://huggingface.co/spaces/mteb/leaderboard>. [Accessed 13-05-2025].

Q.1: What is the AltGDMin algorithm?

A.1: The AltGDMin algorithm is a modification of the Alternating Minimization (AltMin) algorithm, designed for optimization problems where the variables can be split into subsets. In AltGDMin, the slower minimization step in AltMin is replaced by a single gradient descent (GD) step, making it faster and more efficient for partly decoupled problems. It is particularly useful when the cost function is differentiable with respect to one subset of variables and the minimization over the other subset is quick or decoupled.

Q.2: How is AltGDMin different from convex optimization

A.2: AltGDMin differs from convex optimization (specifically convex relaxation) in that it is much faster and more communication-efficient, especially in distributed or federated settings. While convex optimization methods often require many iterations or have high per-iteration computational costs (sometimes cubic in problem size), AltGDMin leverages problem structure (such as decoupling) and uses gradient descent steps to achieve faster convergence. Additionally, AltGDMin is particularly advantageous when the cost function is differentiable with respect to one set of variables and the minimization over the other set is quick.

Q.3: How is AltGDMin different from alternating minimization?

A.3: The AltGDMin algorithm is a faster variant of the Alternating Minimization (AltMin) method that replaces the slower minimization step with a single gradient descent (GD) step. It is designed for problems where the optimization variables "can be split into subsets, with one subsets minimization" being much quicker and the cost function differentiable with respect to the other subset. AltGDMin is particularly useful for partly decoupled problems like robust PCA, robust matrix completion, and robust low-rank compressed sensing.

Q.4: What is the minimization problem of the AltGDMin algorithm for LRMC?

A.4: The minimization problem of the AltGDMin algorithm for LRMC is to update B at each iteration by solving $B_t := \arg \min_B f(U_{t1}, B)$, where $f(U, B)$ is the cost function measuring the difference between the current estimate $X = UB$ and the true matrix X^* . The goal is to minimize $\mathbb{E}[\nabla_U f(U, B)] = (X - X^*)B^\top$, ensuring that $X = UB$ approximates X^* well. This is done alternately with updates to U , using a spectral initialization for U and then alternating minimization steps.

Q.5: What is Newton's method?

A.5: I don't know. Context not relevant.

Tool Results 1: Tool 1 (Agentic RAG) results for AltGDMin [12]

Q.1: What is the strawberry Algorithm?

A.1: The Strawberry Algorithm is a computational method for transcriptome assembly and quantification from RNA-seq data. It uses a step-by-step approach, first assembling transcripts parsimoniously using flow network algorithms, and then quantifying transcript expression based on an insert length model. Strawberry improves computational efficiency by grouping reads into exon bins, reducing data complexity without sacrificing performance.

Q.2: What are the evaluation metrics for strawberry algorithm?

A.2: The evaluation metrics for the Strawberry algorithm include sensitivity and specificity at the nucleotide, exon, intron, and transcript levels. Additionally, the number of correctly assembled transcripts and the correlation with ground truth expression values (such as RPKM compared to NanoString probe counts) are used. These metrics assess both the accuracy of transcript assembly and quantification.

Q.3: What algorithms is the strawberry algorithm compared to?

A.3: The strawberry algorithm is compared to StringTie and Cufflinks, which are two other leading methods for transcript assembly and quantification. It is also mentioned alongside FlipFlop and Traph, which utilize flow network algorithms. However, the primary direct comparisons in performance are with StringTie and Cufflinks.

Q.4: What do the authors of strawberry use expectation maximization (EM) for?

A.4: The authors of strawberry use expectation maximization (EM) to iteratively estimate the parameters θ and β in their model, maximizing one while holding the other constant. Specifically, with β fixed, θ is solved using the EM algorithm, and vice versa. This approach allows them to efficiently optimize their bi-concave likelihood function.

Q.5: What is Newton's method?

A.5: I don't know. Context not relevant.

Tool Results 2: Tool 1 (Agentic RAG) results for Strawberry paper [13]

Problem 1. Input Prompt: A manager needs to assign 3 workers to 3 jobs. Each worker can do any job, but the cost of assigning each worker to each job is different.

	Job 1	Job 2	Job 3
Worker 1	9	2	7
Worker 2	6	4	3
Worker 3	5	8	1

Objective: assign each worker to one unique job so that the total cost is minimized.

Answer: array([0, 1, 2]), array([1, 0, 2]).

Problem 2. A logistics firm is planning to deploy four delivery drones (D1, D2, D3, D4) to deliver urgent packages to four distinct city zones (Z1, Z2, Z3, Z4). However, each drone has a varying level of battery efficiency and signal stability, making them more or less suited to particular zones.

Each zone has unique geographical characteristics (like signal interference, wind patterns, and airspace restrictions), and assigning a drone to a suboptimal zone can result in delayed delivery and higher battery consumption. The estimated cost (in minutes of delay) of assigning each drone to each zone, after factoring in terrain, air traffic, and drone characteristics, is given in the table below:

	Z1	Z2	Z3	Z4
D1	9	2	7	8
D2	6	4	3	7
D3	5	8	1	8
D4	7	6	9	4

The company's objective is to minimize the total delivery delay by assigning each drone to exactly one zone and each zone to exactly one drone. **Additional Constraints:**

- Due to a recent update in drone firmware, Drone D3 should not be assigned to Zone Z2.
- Drone D2 must be assigned to either Zone Z2 or Z3 because of its signal calibration.
- Assignments must be exclusive — no two drones can be assigned to the same zone.

GPT-4.1 Answer: [0,1,2,3],[1,0,2,3] (Incorrect assignment).

GPT-4o Answer: [0,1,2,3], [1,2,0,3] (Correct assignment).

Tool Results 3: Tool 3: Linear Assignment Problem (LAP) Results. Note that the assignment output by GPT-4.1 is incorrect because it assigns drone D2 [index = 1] to Zone 1 [index = 0], which violates the second of three constraints. To understand why, please see the relevant discussion in the results section.