HOW UNDERSPECIFIED PROMPTS SHAPE TOOL-CALLING LLM AGENTS IN SCIENTIFIC WORKFLOWS

Ahmed Osama Mohamed Muharram

A THESIS

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Master of Science in Engineering

2026



Chris Callison-Burch, Professor of Computer and Information Science
Supervisor of Thesis



Eric Wong, Assistant Professor of Computer and Information Science
Reader of Thesis



Swapneel Sheth, Practice Associate Professor of Computer and Information Science
CIS/MSE Program Chair

HOW UNDERSPECIFIED PROMPTS SHAPE TOOL-CALLING LLM AGENTS IN SCIENTIFIC WORKFLOWS

© COPYRIGHT

2026

Ahmed Osama Mohamed Muharram

This work is licensed under the
Creative Commons Attribution
4.0 International
License

To view a copy of this license, visit

https://creativecommons.org/licenses/by/4.0/

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to the people who supported and guided me throughout my studies and the completion of this thesis at the University of Pennsylvania.

First and foremost, I am deeply grateful to Prof. Chris Callison-Burch for his exceptional guidance, mentorship, and encouragement throughout this project. His support went far beyond regular supervision: he helped me refine the research direction, consistently challenged me to raise the quality and clarity of my work, and connected me with collaborators and resources that were essential to bringing this thesis to completion. I am especially thankful for his thoughtful feedback and steady confidence in this project.

I am grateful to Delip Rao and Andrew Zhu for their help with the LAMINA case study. Delip shared assessments that strengthened my understanding of benchmarking and evaluation, while Andrew provided guidance on Kani and tool-calling that helped shape the system's development.

I would like to thank Prof. David S. Roos for his guidance in shaping PathFinder's scope and for connecting me with the VEuPathDB team. I am sincerely grateful to the VEuPathDB maintainers and contributors for their time, meetings, and insights, and for helping ground this project in practical needs and real use cases.

I also thank Prof. Eric Wong for serving as my thesis reader and for his careful review and constructive comments on the final drafts. His feedback helped improve the organization and presentation of this work.

Finally, I am profoundly grateful to my family for their unwavering love, patience, and support throughout this journey; for believing in me through the most stressful and uncertain moments. I am also thankful to my friends, both near and far, for their encouragement and constant reassurance during the most demanding stages of this work.

ABSTRACT

Tool-calling LLM agents can translate natural-language instructions into structured invocations of scientific APIs and databases, but natural language is inherently underspecified: researchers routinely omit dataset identifiers, parameter thresholds, and success criteria that downstream tools require. When this happens, the agent must infer or default missing tool parameters. This thesis studies how prompt underspecification shapes the behavior of such agents through two case studies: LAMINA, a 50-tool materials science agent evaluated on 37 feasibility claims across three models, and PathFinder, a 44-tool pathogen genomics agent evaluated on 35 gold-standard search strategies at three vagueness levels across two models and 420 total runs. Both are built on the Kani framework and are open source.

Prompt underspecification was the dominant source of output instability in both case studies, outweighing the effects of model size, routing policy, and task complexity within the ranges evaluated here. In LAMINA, vague claims induce a flip rate of 0.43, meaning that alternate plausible interpretations of the same claim often produce different final labels. In PathFinder, removing dataset-level specification collapses gene-set F1 from 0.664 to 0.227, and further vagueness reduces it to 0.158. The instability localizes to the natural-language-to-tool-invocation mapping: tools function correctly once given precise inputs, but the agent's translation from the user's prompt into tool choices and arguments is unreliable, and in multi-step tasks these early mistakes propagate through later operations. Within the range of models evaluated here, model scaling does not currently compensate. We derive four design recommendations: disambiguation protocols, catalog grounding, incremental validation, and replicated evaluation designs.

TABLE OF CONTENTS

ACKNOWLEDGEMENT iii

ABSTRACT iv

LIST OF TABLES ix

LIST OF ILLUSTRATIONS xi

TERMINOLOGY xv

0.1 Common to LAMINA and PathFinder xv

0.2 For LAMINA xvi

0.3 For PathFinder xxi

CHAPTER 1 : Introduction 1

1.1 Motivation 1

1.2 Problem Setting 2

1.3 Taxonomy of Underspecification 3

1.4 Research Questions 5

1.5 Contributions 6

1.6 Background and Related Work 7

1.7 Thesis Organization 10

CHAPTER 2 : LAMINA: Agent Design for Materials Feasibility Assessment . . . 12

2.1 Overview and Scientific Context 12

2.2 Target Task and Domain Scope 13

2.3 Design Goals 14
2.4 Architecture and the Kani Framework 15
2.5 Tooling Subsystems (50 Functions) 16
2.6 Default Routing Policy and Guardrails 19
2.7 Positioning Relative to Prior Systems 20
2.8 Chapter Summary 21

CHAPTER 3 : LAMINA: Experiments on Vagueness and Underspecification . . . 22
3.1 Benchmark, Claim Partition, and Research Questions 22
3.2 Experimental Setup 23
3.3 Results 28
3.4 Failure Modes and Underspecification Effects 34
3.5 Discussion and Takeaways 39
3.6 Chapter Summary 41

CHAPTER 4 : PathFinder: Agent Design for VEuPathDB Strategy Construction . 42
4.1 Overview and Scientific Context 42
4.2 Target Task and Strategy Abstraction 43
4.3 Design Goals 46
4.4 Architecture and the Kani Framework 46
4.5 Tool Registry 48
4.6 Search Discovery and Semantic Retrieval 49
4.7 How PathFinder Builds a Strategy 50
4.8 Chapter Summary 51

CHAPTER 5 : PathFinder: Experiments on Vagueness in Strategy Construction . . 52

5.1 Gold Corpus, Vagueness Protocol, and Experimental Questions 52
5.2 Experimental Setup 56
5.3 Results 60
5.4 Failure Modes and Underspecification Effects 67
5.5 Discussion 69
5.6 Chapter Summary 71

CHAPTER 6 : Discussion and Conclusion 72
6.1 Cross-Case Synthesis 72
6.2 Common Failure Patterns 74
6.3 Design Recommendations 75
6.4 Shared Limitations 78
6.5 Future Work 79
6.6 Conclusion 81

APPENDIX A : LAMINA Supplementary Materials 84
A.1 System Execution Examples 84
A.2 Example Drop-4 Claim Trace: Silver Bulk Modulus Feasibility Claim 99
A.3 Tool Registry and Function Reference 103
A.4 Example Failure Traces 113
A.5 Drop-4 Claims: Precise vs. Vague 116

APPENDIX B : PathFinder Supplementary Materials 122
B.1 Complete Tool Registry 122
B.2 Gold-Standard Strategy Corpus 123
B.3 WDK Search Types Exercised By The Gold Corpus 125

BIBLIOGRAPHY 127

LIST OF TABLES

TABLE 2.1 : Tool families in the LAMINA tool registry. Counts refer to distinct functions. A full function-level reference is provided in Appendix A. 18

TABLE 2.2 : Comparison of LAMINA with prior materials science tool-augmented systems (Zhang et al., 2024; Jia et al., 2024; Chaudhari et al., 2025; Liu et al., 2025a). 20

TABLE 3.1 : Agreement and efficiency across models and tool settings on Drop-4 ($N = 37$). $\Delta\kappa$ is computed relative to the corresponding no-tools configuration. 28

TABLE 3.2 : Summary of pre-registered hypothesis verdicts on Drop-4. Only H2b (routing eliminates tool misses) and H3a (ambiguity induces flip rate instability) are formally supported. 34

TABLE 4.1 : Core strategy-construction tools exposed by PathFinder, grouped by category. These 19 tools cover discovery, graph mutation, execution, inspection, and external research. Additional utilities are part of the broader 44-tool registry but omitted here for brevity. . . 49

TABLE 5.1 : Performance across vagueness levels on both models ($N = 35$ strategies per condition; mini values are means of 3 replicates). 95% bootstrap CIs in brackets. All metrics are higher-is-better. Perfect = fraction of runs with $F_1 = 1.0$. 60

TABLE 6.1 : Side-by-side comparison of the two case studies. 72

TABLE A.1 : Precise Drop-4 claims, grouped by domain and claim type. Full verbatim claim text appears below. 116

TABLE A.2 : Vague Drop-4 claims, grouped by ambiguity family. Full mapping details and verbatim claim text are provided in the benchmark release, and the full verbatim claim text appears below. 118

TABLE B.1 : Complete tool registry (44 tools). During evaluation, all catalog discovery tools (9), strategy mutation (5), execution (1), sample results (1 of 2), inspection (1), and research (2) are enabled (19 tools total, Table 4.1). The remaining 25 tools support production features not relevant to strategy construction. 122

TABLE B.2 : The 35 gold-standard strategies, grouped by complexity bin and sorted by AST node count within each bin. †Public user strategies from VEuPathDB's shared repository. Database coverage spans 8 sites: PlasmoDB (12), FungiDB (7), ToxoDB (4), TriTrypDB (4), CryptoDB (3), VectorBase (3), GiardiaDB (1), MicrosporidiaDB (1). 123

TABLE B.3 : WDK search types exercised by the 35 gold strategies. *Strategies* = number of gold strategies using this search type; *Variants* = number of dataset-specific concrete search names (e.g., the 13 RNA-Seq variants correspond to 13 organism-specific datasets). Types marked with * have dataset-specific suffixes in the WDK API. Total: 31 base types, 44 concrete search names. 125

FIGURE 2.1 : A sample query to generate a phase diagram using LAMINA. The sidebar shows the tool call trace, including arguments and return values. 16

FIGURE 2.2 : Backend architecture and tool orchestration. FastAPI hosts a Kani-based controller that selects tools, calls external services (CALPHAD, Materials Project, CHGNet, etc.), and emits SSE traces for each interaction. 17

FIGURE 3.1 : H1: Tools vs. no-tools on Drop-4. 28

FIGURE 3.2 : H2: Routing quality for `gpt-4o` on Drop-4. 29

FIGURE 3.3 : H3: Flip rates and agreement metrics on Drop-4. 31

FIGURE 3.4 : Range of possible QWK on vague claims for best vs. worst numeric remappings of ambiguous quantifiers. 32

FIGURE 3.5 : Agreement (QWK) for `gpt-4o` with tools under single-call vs. multi-call settings. Arrow annotation shows $\Delta\kappa$ and its 95% confidence interval. 33

FIGURE 3.6 : H5: Model size and diminishing returns with tools on Drop-4. . 34

FIGURE 3.7 : Ambiguity-induced flips for the Al-Zn eutectic claim (*"Al and Zn form an eutectic at ∼Al15Zn75"*). Three interpretations of "$\sim$" (a tight $\pm1\%$ window, a moderate $\pm5\%$ window, and a loose $\pm10\%$ window around Al15Zn75) yield predicted labels of -2, $+2$, and $+2$, respectively, for a gold label of $+2$. 36

FIGURE 3.8 : Ambiguity-induced flips for the Fe_2O_3 Al-doping claim ("Someone made a stronger magnet by doping α-Fe_2O_3 with Al"). Interpreting "stronger magnet" as higher M_s, higher coercivity, higher pull force, or higher remanence yields predicted labels of -1, $+1$, -2, and -2, respectively, for a gold label of $+2$. 37

FIGURE 4.1 : A concrete example strategy for finding likely membrane transporters in *P. vivax* P01. Two leaf searches are combined with an intersect operator: one filters genes by transmembrane domain count (5–12 domains), and the other filters genes by the GO term *transmembrane transporter activity* (GO:0022857). The final result contains 13 genes. 44

FIGURE 4.2 : PathFinder's interface during strategy construction for the `pvivax_tm_transporter` strategy. The conversation panel shows the agent's reasoning and proposed strategy, while the bottom bar displays the minimized strategy graph: a `GenesByGoTerm` leaf node and a `GenesByTransmembraneDomains` leaf node joined by an intersection operator. 45

FIGURE 4.3 : PathFinder's five-layer architecture. 47

FIGURE 5.1 : Gold-standard AST for the `pvivax_tm_transporter` strategy (complexity 3). The root is an intersect node with two leaf searches: one filtering by transmembrane domain count (5–12) and one by GO term *transmembrane transporter activity* (GO:0022857). All parameter values are strings, matching the WDK REST API convention. This strategy yields 13 *P. vivax* P01 genes. 53

FIGURE 5.2 : Gene-set F_1 by prompt vagueness (`gpt-4.1`, $N = 35$). Error bars show 95% bootstrap CIs. The precise–moderate gap is the dominant transition. 61

FIGURE 5.3 : Degradation of all strategy components under increasing vagueness (`gpt-4.1`, $N = 35$). Search accuracy, parameter fidelity, and operator accuracy all decline; search accuracy is the strongest predictor of functional accuracy ($\rho = 0.574$). 62

FIGURE 5.4 : Gene-set flip rate on `gpt-4.1-mini` ($N = 35$ strategies, 3 replicates per condition). Even for precise prompts, 56% of replicate pairs produce different gene sets. 63

FIGURE 5.5 : H2: Complexity and vagueness compound (`gpt-4.1`, $N = 35$). (a) Both factors independently degrade performance; their combination produces near-total failure. (b) Most strategies drop steeply from precise to moderate, with several large strategies near zero across all conditions. 65

FIGURE 5.6 : Gene-set F_1 for `gpt-4.1` ($N = 35$) vs. `gpt-4.1-mini` (mean of 3 replicates, $N = 35$) at each vagueness level. The vagueness gradient replicates on the smaller model, with the gap largest for precise prompts. 66

FIGURE 5.7 : Two failure modes from real evaluation runs (`gpt-4.1`). (a) Dataset selection error: the moderate prompt causes the agent to select a different proteomics experiment (Wiedner vs. Afum-CEA10) for a 3-node strategy, producing zero gene overlap despite correct tree structure. The vague prompt defaults to the correct dataset ($F_1 = 0.864$). (b) Parameter-level error propagation: the agent builds the identical 7-node tree as the gold (SA $= 1.0$, PFS $= 0.95$), but a single wildcard substitution in the GO term field (`"*kinase*"` instead of `"N/A"`) inflates the gene set from 71 to 621, producing 550 false positives ($F_1 = 0.205$). . . 67

TERMINOLOGY

0.1. Common to LAMINA and PathFinder

API (Application Programming Interface) A structured way for one software system to communicate with another.

FastAPI A Python framework used to build web services and APIs.

Flip Rate (FR) A measure of how often a model's output changes across alternate interpretations or repeated runs.

Frontend The part of a software system that users directly interact with, such as the chat interface.

Kani A lightweight framework for building tool-calling LLM applications, used in this thesis to manage chat history, expose tools, and handle the core tool-calling loop.

LLM (Large Language Model) An AI model trained on large amounts of text that can generate and analyze language.

React A JavaScript library for building interactive web interfaces; not to be confused with ReAct, the prompting framework for interleaving reasoning and actions.

REST API A web-based interface that allows programs to access and manipulate data using standard HTTP requests.

Server-Sent Events (SSE) A web communication method that allows a server to continuously stream updates to a client.

Streaming trace A live record of the tool calls, arguments, and outputs produced while the system is working.

Tool registry The collection of callable functions available to the AI system.

0.2. For LAMINA

Anode The electrode in a battery where oxidation occurs; in lithium-ion batteries, it is the side that stores and releases lithium during charging and discharging.

CALPHAD (CALculation of PHAse Diagrams) A computational method used to predict which phases of a material are stable under different temperatures and compositions.

Capacity In electrochemistry, the amount of electric charge a battery material can store, usually reported relative to mass.

CHGNet Short for *Crystal Hamiltonian Graph neural Network*, a machine-learning model used in materials science to approximate results that would otherwise require more expensive quantum-mechanical calculations.

Coercivity A measure of how strongly a magnetic material resists being demagnetized.

Composition-temperature space The range of material compositions and temperatures over which a material's behavior is studied.

Convex hull A stability analysis tool in materials science used to determine whether a material is thermodynamically stable compared with other possible compounds.

Crystal symmetry The geometric pattern or repetition in the arrangement of atoms in a crystal.

Cuprate A copper-oxide-based material, often studied because some cuprates exhibit high-temperature superconductivity.

Database validity range The range of inputs, such as temperature or composition, for which a scientific database is considered reliable.

Defect formation energy The energy required to create an imperfection in a crystal, such as a missing atom or substituted atom.

DFT (Density Functional Theory) A quantum-mechanical computational method used to estimate the properties of materials at the atomic scale.

Drop-4 A 37-claim benchmark from DARPA's Scientific Feasibility (SciFy) program used to evaluate whether a system can judge the feasibility of natural-language materials science claims on an ordinal scale from strongly infeasible to strongly feasible.

Dopant An element intentionally added in small amounts to a material to change its properties.

Electrode A conductive part of a battery or electrochemical device where reactions occur.

Elastic properties Properties that describe how a material deforms and returns to shape when forces are applied.

Equilibrium A state in which a material system is stable under given conditions and no net change is expected over time.

Eutectic A specific mixture of components that melts or solidifies at a single, unusually low temperature compared with nearby compositions.

Feasibility rating The score used in the benchmark to judge how scientifically plausible a claim is, from strongly infeasible to strongly feasible.

Jahn–Teller distortion A structural distortion in certain crystals caused by the electronic structure of the atoms, often affecting stability and properties.

LAMINA Short for *LLM-Assisted Material INformatics and Analysis*, the materials-science system described in this chapter.

Lattice parameter A number that describes the size of a crystal's repeating unit cell.

Li-ion Short for lithium-ion, a common rechargeable battery chemistry.

Lithiation The process by which lithium atoms are inserted into a material.

MAE (Mean Absolute Error) A metric that measures, on average, how far predictions are from the correct values.

Materials informatics The use of computation, databases, and AI to study and design materials.

Materials Project (MP) A large scientific database containing computed properties for many inorganic materials.

Microstructure The internal structure of a material, including its phases and their arrangement, often visible at small scales.

NEB (Nudged Elastic Band) A computational method for estimating the energy barrier between two atomic configurations, often used for diffusion studies.

Ordinal scale A ranking scale in which categories have an order, but the distance between categories is not necessarily uniform.

Phase A physically distinct form of matter within a material, such as a solid solution or intermetallic compound.

Phase diagram A chart showing which phases are stable at different temperatures, pressures, or compositions.

Phase fraction The proportion of a material that exists as a particular phase.

Phase stability Whether a phase is expected to exist under specific conditions such as temperature and composition.

Policy router A set of hand-designed rules that helps decide which tool family should be used for a claim.

Pycalphad A Python software library for performing CALPHAD-based thermodynamic calculations.

QWK (Quadratic Weighted Kappa) A metric for measuring agreement on ordered labels, giving larger penalties when predictions are farther from the correct answer.

QWFR (Quadratic-Weighted Flip Rate) A version of flip rate that also accounts for how large the label changes are, not just whether they changed.

Remanence The amount of magnetization left in a material after an external magnetic field is removed.

Routing The process of choosing which computational tools an AI system should use for a given question.

Saturation magnetization (M_s) The maximum magnetization a material can reach in an applied magnetic field.

Scheil–Gulliver solidification A model used to estimate how an alloy solidifies when diffusion in the solid is limited.

SciFy DARPA's Scientific Feasibility program, which evaluates whether technical claims can be checked against tools, models, and trusted data.

Semantic ambiguity Uncertainty caused by unclear meaning in a phrase, such as whether "stronger magnet" refers to magnetization, coercivity, or another property.

Site preference The tendency of a dopant or defect to occupy one atomic position in a crystal rather than another.

Solubility The extent to which one element or compound can dissolve into another material.

Space group A formal description of the symmetry of a crystal structure.

Surrogate model A faster computational model used to approximate the results of a slower, more expensive method.

TDB (Thermodynamic Database) A file or database containing parameters used for CALPHAD thermodynamic calculations.

Thermodynamics The study of energy, heat, and equilibrium in physical systems; in materials science, it helps predict which phases are stable.

Tool miss-rate The fraction of cases in which the system fails to call a tool family that is needed for the task.

Vegard's law An approximate rule stating that the lattice parameter of an alloy often changes roughly linearly with composition.

Voltage-capacity curve A graph showing how a battery material's voltage changes as charge is added or removed.

0.3. For PathFinder

Abstract syntax tree (AST) A tree-structured representation of a strategy in which each node stores a search, transform, or combination step and its parameters.

ANOVA (Analysis of Variance) A statistical method that separates variation into between-group and within-group components. In this thesis, a one-way ANOVA is used when computing the intraclass correlation coefficient (ICC).

BCa confidence interval Short for *bias-corrected and accelerated* bootstrap confidence interval, a bootstrap-based interval estimate that adjusts for bias and skewness in the sampling distribution.

Boolean operator A logical operation used to combine search results, such as `INTERSECT`, `UNION`, or `MINUS`.

Catalog discovery The process of finding which searches are available in the database and what inputs they require.

Complexity bin A grouping of strategies by size, usually based on how many nodes appear in the strategy tree.

Corpus A curated collection of examples used for evaluation or analysis.

CQRS (Command Query Responsibility Segregation) A software design pattern that separates operations that modify data from operations that read data.

Dataset A specific collection of experimental or computational data, such as an RNA-seq study or proteomics experiment.

Expression profile A measurement of how strongly genes are turned on or off under certain biological conditions.

F_1 score A metric that summarizes overlap between two sets by balancing precision and recall. In PathFinder, it is used mainly to compare predicted and gold gene sets.

Gene Ontology (GO) A standardized vocabulary used to describe gene functions, biological processes, and cellular locations.

Gene set A group of genes returned by a search or strategy.

Genomics The study of genes and whole genomes, including their structure, function, and variation.

Gold strategy An expert-authored reference strategy used as the correct target during evaluation.

Gold strategy corpus The collection of 35 reference strategies used to evaluate PathFinder.

ICC (Intraclass Correlation Coefficient) A statistic used to measure how consistent repeated runs are with one another.

Leaf search A basic search step at the edge of a strategy tree that does not depend on previous steps.

Linear sum assignment An optimization procedure that finds the best one-to-one matching between two sets of items. In PathFinder, it is used to align predicted and gold strategy nodes when computing search accuracy.

Node A single unit in a strategy tree, such as a search, transform, or Boolean combination.

NTED (Normalized Tree Edit Distance) A measure of how different two tree structures are after accounting for the edits needed to transform one into the other.

Omics A broad term for large-scale biological data types, such as genomics, transcriptomics, or proteomics.

Operator accuracy A measure of whether the system chose the correct Boolean operation, such as intersection or union, at each combination step.

Ortholog A gene in one species that evolved from a corresponding gene in another species and often has a similar function.

Parameter A specific input value for a search, such as an organism name, dataset, threshold, or GO term.

Parameter Fidelity Score (PFS) A measure of how accurately the system reproduces the parameter values used in the gold strategy.

PathFinder Short for *Pathogen Finder*, the tool-augmented LLM system described in this chapter that builds genomics search strategies from natural-language prompts.

Precision In set comparison, the fraction of predicted items that are correct.

Prompt vagueness The degree to which a user request leaves out important technical details, such as exact datasets, thresholds, or search structure.

Proteomics The study of proteins, including their abundance, structure, and function.

Public user strategy A strategy shared by a researcher in VEuPathDB's public repository.

Recall In set comparison, the fraction of correct items that were successfully retrieved.

Record type The kind of object a search returns, such as genes or transcripts.

RNA-seq Short for RNA sequencing, a method used to measure gene activity by sequencing RNA molecules.

Search accuracy A measure of how often the system selected the same search steps as the gold strategy.

Semantic search A search method that retrieves relevant results based on meaning rather than exact keyword matches alone.

Single-turn interaction An interaction in which the system must complete the task from one prompt without asking follow-up questions.

SNP (Single Nucleotide Polymorphism) A variation at a single DNA position among individuals or strains.

Spearman correlation A statistical measure of how strongly two ranked quantities are associated.

Strategy In VEuPathDB and PathFinder, a multi-step query plan that combines searches and operations to produce a final gene set.

Strategy graph The internal working representation of a strategy as connected steps that can be edited, validated, and executed.

Transform A strategy step that converts one set of results into another, such as mapping genes to orthologs in another species.

Transcript An RNA copy produced from a gene; transcript-level searches focus on these RNA products rather than the gene as an abstract unit.

VEuPathDB (Eukaryotic Pathogen, Vector, and Host Informatics Resources) A consortium of related biological databases covering eukaryotic pathogens, vectors, and hosts.

WDK (Strategies Web Development Kit) The search framework used by VEuPathDB to build and run multi-step strategies.

CHAPTER 1 : Introduction

1.1. Motivation

Scientific research increasingly depends on complex, domain-specific computational toolchains. A malaria researcher studying drug resistance might construct a multi-step search strategy in VEuPathDB (the Eukaryotic Pathogen, Vector, and Host Informatics Resources) (Amos et al., 2022) that combines gene expression profiles, protein domain annotations, and ortholog relationships across species. A materials scientist evaluating a candidate alloy might invoke CALPHAD (CALculation of PHAse Diagrams) equilibrium solvers (Otis and Liu, 2017), first-principles databases, and neural surrogates to assess phase stability. These workflows are powerful, but they impose a substantial translation burden: the researcher must convert a scientific question into the precise, structured inputs that each tool requires, a process that demands deep familiarity with parameter schemas, data availability, and platform-specific conventions.

Recent advances in large language models (LLMs) have opened a new path. LLMs can now use external tools autonomously: given a natural-language instruction, the model emits a structured request to call an API, database, or computational library, receives the returned result, and continues reasoning over it (Schick et al., 2023; Yao et al., 2023; Qin et al., 2023). The model itself does not execute the underlying code; it requests that an external function or service run and then reasons over the output. This capability has given rise to LLM agents that observe, plan, and act within external environments, and scientific toolchains are a natural deployment target.

The promise of such agents is that researchers could describe their questions in natural language and have the agent handle the translation into tool calls. Natural language, however, is inherently underspecified. A request for "genes upregulated in the sexual stage" omits the organism, the RNA sequencing (RNA-seq) experiment, the fold-change thresh-

old, and the statistical cutoff. These are structured parameters that the downstream search tools available to the agent require. A claim that "a few percent zinc stabilizes the τ-phase in an Al–Mg matrix" leaves open whether "a few percent" means 2%, 3%, or 5%, and the thermodynamic answer differs across those values because the downstream tools require a specific composition. Such omissions reflect the default register of scientific communication, in which researchers rely on shared context and disciplinary convention to fill in the details. An LLM agent, lacking that shared context, must resolve the ambiguity on its own: it may infer a value from domain priors, choose a default, or guess incorrectly.

This thesis studies a single empirical question: **how does prompt underspecification shape the behavior of tool-calling LLM agents in scientific workflows?** We address this question through two case studies in different scientific domains, materials science and eukaryotic pathogen genomics, using purpose-built agent systems that interact with real, production-grade toolchains.

1.2. Problem Setting

We consider the following setting. A user provides a natural-language prompt p describing a scientific task (e.g., a feasibility judgment, a database query). An LLM agent $\mathcal{A}$ interprets p and produces a sequence of tool invocations $t_1, t_2, \ldots, t_k$, where each t_i consists of a tool name and structured arguments. The tools may be deterministic (a database query) or stochastic (a simulation with random initial conditions), but in the systems we study they are predominantly deterministic: the dominant source of variability is the agent's choice of tools and arguments, not the tool execution itself. The agent uses tool results to produce a final output y (a label, a structured artifact, a gene set).

The central variable is the specificity of p. A fully specified prompt contains all the information needed to determine the correct tool invocations uniquely. An underspecified

prompt omits some of that information, forcing the agent to infer, guess, or default. We measure how the quality of y degrades as p becomes less specific, and in which stage of the pipeline the degradation manifests: tool selection, parameterization, or composition. Here, tool selection refers to choosing the appropriate tool or tool family, parameterization refers to supplying the correct structured arguments, and composition refers to correctly chaining tool outputs into later tool inputs or later reasoning steps.

We focus on narrow, closed-domain scientific tasks with expert-authored reference outputs, so that deviations from the target can be measured precisely. We do not study open-ended dialogue, creative generation, or tasks without a ground truth.

We impose two additional constraints on the evaluation protocol. First, all evaluations use a single user prompt: the agent receives one prompt and must produce a complete output without follow-up interaction. Second, all prompts explicitly instruct the agent not to ask clarifying questions. We do not separately score attempted clarification behavior; the evaluation measures only whether the agent completes the task under this no-clarification constraint. Both systems were designed for multi-turn use, in which the agent actively solicits missing details from the researcher and supports iterative refinement of the output. Our results therefore represent a lower bound on achievable accuracy, and multi-turn interaction would likely improve performance, particularly on vague prompts.

1.3. Taxonomy of Underspecification

To organize observations across the two case studies, we identify five categories of information that scientific prompts commonly omit. The taxonomy's purpose here is to provide a consistent vocabulary for the failure modes reported in Chapters 3 and 5.

In a tool-calling rollout, the user provides a natural-language prompt, the model selects from a developer-provided set of tools, emits a structured invocation, receives the tool's

output, and may repeat this process over multiple steps before producing a final answer. Underspecification can therefore affect not only the final response, but also each intermediate stage of tool choice, argument formation, and multi-step composition.

1. **Missing objectives and success criteria.** The prompt states a topic but not what constitutes a satisfactory answer. A researcher who asks "Tell me about kinase genes in P. falciparum" has not specified whether the goal is to enumerate them, analyze their expression, or identify drug targets among them. Note that the agents retain access to literature and web search tools throughout evaluation and can use them to answer general scientific questions. The evaluation in this thesis, however, measures a different capability: whether the agent constructs the correct structured artifact (a search strategy in PathFinder, a feasibility label in LAMINA), which requires selecting specific tools, setting precise parameter values, and composing them correctly.

2. **Missing constraints and operating assumptions.** The prompt omits boundary conditions that narrow the solution space. "Find genes upregulated in the sexual stage" leaves open the organism, the life-cycle comparison, and the fold-change threshold.

3. **Ambiguous entities and references.** The prompt uses terms that could resolve to multiple distinct objects. "A stronger magnet" could refer to higher saturation magnetization, higher coercivity, or higher pull force; the appropriate tool invocation differs in each case.

4. **Ambiguous quantities and thresholds.** The prompt uses vague quantifiers where tools require precise numbers. "A few percent zinc" maps to 2%, 3%, or 5%; "$\sim$100 GPa" maps to a tolerance of $\pm$5, $\pm$10, or $\pm$20.

5. **Ambiguous tool choice and data sources.** The prompt describes a data type but not the specific tool, dataset, or experiment. "Genes with high expression in gameto-

cytes" could refer to any of several RNA-Seq experiments for the same organism, and these experiments may share few or no genes in their result sets. In real workflows, expert users often omit these details because they may already know the relevant dataset or experiment from the literature, and because VEuPathDB's search interface helps them browse and refine candidate searches. An agent, however, still has to resolve that omission into a concrete tool call.

Categories 1-2 concern what the user wants; categories 3–5 concern how the user describes it. In practice, a single prompt often exhibits multiple categories simultaneously: a vague biological question (category 1) typically also omits the dataset (category 5) and the threshold (category 4).

Both case studies operationalize this taxonomy by constructing prompts at controlled vagueness levels. In LAMINA, claims are partitioned into *precise* (containing explicit numbers) and *vague* (containing ambiguous quantifiers), primarily exercising categories 3 and 4. In PathFinder, prompts are authored at three levels (*precise*, *moderate*, *vague*), with the moderate level removing category-5 information (dataset names) and the vague level additionally removing categories 1 and 2 (the scientific goal is stated, but without specifics).

1.4. Research Questions

RQ1. Does prompt underspecification degrade the output quality of tool-calling LLM agents in scientific workflows? How large is the effect relative to other factors such as model scale, task complexity, and routing policy?

RQ2. Which components of tool use (selection, parameterization, composition) are most sensitive to underspecification? Is there a single bottleneck, or does degradation spread across all components?

RQ3. Does underspecification affect *reliability* (run-to-run consistency) in addition to *accuracy*? Can a researcher expect reproducible results when re-running the same vague prompt?

RQ4. Do these effects generalize across scientific domains, task types, and model scales, or are they artifacts of a specific system or benchmark?

1.5. Contributions

This thesis makes four contributions:

1. **Two scientific agent systems:** We present LAMINA[1], a tool-augmented system for materials science feasibility assessment with 50 domain tools (Chapter 2), and PathFinder[2], a system for constructing multi-step genomics search strategies over VEuPathDB's 14 pathogen databases with 44 tools (Chapter 4). Both are built on the Kani framework (Zhu et al., 2023) and are open-source.

2. **Controlled empirical studies of underspecification:** We evaluate both systems under controlled vagueness conditions: 37 materials science claims (precise vs. vague) across 3 models and 3 routing configurations in LAMINA, and 35 gold-standard genomics strategies at 3 vagueness levels across 2 models (420 total runs) in PathFinder.

3. **Cross-domain evidence that, among the factors investigated in this thesis (prompt underspecification, model size, routing policy, and task complexity), underspecification is the dominant source of output instability.** In both systems, the instability consistently localizes to the natural-language-to-tool-invocation mapping rather than to tool execution.

[1] Available at `https://github.com/ahmedomuharram/LAMINA`
[2] Available at `https://github.com/ahmedomuharram/PathFinder`

4. **Design recommendations:** We derive actionable principles for building scientific tool-calling agents, including disambiguation protocols, catalog grounding, incremental validation, and replicated evaluation designs (Chapter 6).

1.6. Background and Related Work

1.6.1. Tool-Calling LLMs and Agent Architectures

The idea that language models can learn to use external tools has progressed rapidly. Earlier work explored narrower forms of tool use and argument selection, including *Giving BERT a Calculator* (Andor et al., 2019) and TALM (Tool Augmented Language Models) (Parisi et al., 2022). These systems did not yet frame the problem as general-purpose agentic tool use, but they helped establish the core idea that language models can identify when external operations are needed and supply the arguments required to invoke them. Building on this line of work, Toolformer (Schick et al., 2023) showed that LLMs can be trained in a self-supervised manner to insert API calls into their own completions, deciding autonomously when a calculator, search engine, or translation system would improve the response. ReAct (Yao et al., 2023) then interleaved chain-of-thought reasoning with tool actions in a single prompt, demonstrating that explicit reasoning traces improve both task performance and interpretability. ToolLLM (Qin et al., 2023) extended this direction to 16,000+ real-world APIs, training models to navigate complex API documentation and compose multi-step tool sequences.

These developments naturally extend into agentic settings, where the model must operate over longer horizons, maintain state across multiple steps, and compose intermediate tool outputs rather than issuing a single isolated call. Subsequent work introduced more sophisticated agent architectures. Reflexion (Shinn et al., 2023) added verbal self-reflection, where the agent critiques its own prior attempts and incorporates that linguistic feedback

into later trials. Voyager (Wang et al., 2023) demonstrated open-ended skill acquisition in Minecraft, where an LLM-powered agent continuously explores, generates executable code as reusable skills, and stores them in a persistent library that compounds its capabilities over time.

A recurring finding across this literature is that tool use is brittle. Sun et al. (2024) introduced a framework for detecting "silent" tool errors and found that models struggle to identify when a tool has returned incorrect results, often propagating faulty outputs into their final answers without verification. Yu et al. (2025) found that in chemistry problem solving, tool-augmented agents do not consistently outperform base LLMs without tools; on general chemistry questions (e.g., exam-style problems requiring broad reasoning), tool augmentation introduced additional cognitive load that hindered rather than helped performance. These observations motivate our focus on the natural-language-to-tool-invocation mapping as the critical failure point.

1.6.2. LLM Agents in Scientific Workflows

Scientific domains have been early adopters of tool-augmented LLMs, in part because many scientific toolchains already expose machine-callable interfaces.

In chemistry, ChemCrow (Bran et al., 2023) augmented GPT-4 with 18 expert-designed tools to accomplish tasks across organic synthesis, drug discovery, and materials design. Yu et al. (2025) subsequently evaluated tool-augmented chemistry agents across specialized tasks (molecular manipulation, synthesis prediction) and general exam-style questions, finding that tools yield substantial gains on specialized tasks but do not consistently help on general questions where broad chemical reasoning is required.

In materials science, several systems have been proposed. HoneyComb (Zhang et al., 2024) built a flexible LLM-based agent system backed by a materials knowledge base. LLMatDesign (Jia et al., 2024) applied LLM agents to autonomous materials discovery.

MatSciAgent (Chaudhari et al., 2025) introduced modular multi-task agents for computational materials science. MatTools (Liu et al., 2025a) provided a benchmark for evaluating materials science tool use. LAMINA (Chapter 2) extends this line of work by combining CALPHAD thermodynamic tools, DFT databases, and neural surrogates into a single agent, and is the first such system to study how linguistic ambiguity affects tool-calling reliability.

In genomics and bioinformatics, tool-augmented agent systems are still emerging rather than absent. VEuPathDB's strategy system (Fischer et al., 2011; Amos et al., 2022) provides a rich, multi-step search interface that has been used interactively by thousands of researchers across 14 pathogen, vector, and host databases. At the same time, adjacent LLM-based work has begun to appear in bioinformatics and computational biology, including systems for automated single-cell analysis, large-scale omics copilot workflows, biomedical knowledge retrieval, and end-to-end bioinformatics assistance (Xiao et al., 2024; Liu et al., 2025b; Bi et al., 2025; Mehandru et al., 2025); recent reviews also document a growing broader literature on biological and bioinformatics LLM agents (Yang et al., 2025; Dip et al., 2026). To our knowledge, however, no prior published work has studied LLM-driven construction of VEuPathDB strategies or examined how prompt vagueness affects strategy-construction fidelity in this specific setting. PathFinder (Chapter 4) addresses this gap.

1.6.3. Prompt Vagueness and Underspecification

The study of how input specificity affects LLM behavior spans several sub-literatures.

In instruction following, researchers have documented that LLMs are sensitive to prompt formatting and phrasing. Sclar et al. (2024) showed that meaning-preserving formatting changes (e.g., delimiter choice, whitespace, label wording) can shift few-shot accuracy by up to 76 percentage points. Lu et al. (2022) demonstrated that, at least in older models, the ordering of few-shot examples alone can cause performance to swing between near state-of-the-art and random-guess levels. These findings establish that surface-level prompt

variation has large effects on LLM outputs even when the underlying task is held constant.

In agent benchmarking, Liu et al. (2024) evaluated LLMs as agents across eight diverse interactive environments and found substantial capability gaps between top commercial and many open-source model families. Although leading commercial models showed comparatively strong agent performance, the benchmark identified poor long-term reasoning, decision-making, and instruction following as major obstacles to reliable agent behavior.

Most prior work, however, treats underspecification as a confound to be controlled rather than a variable to be studied. Benchmarks typically use fully specified prompts, and evaluation protocols assume that the correct tool invocation is deterministic given the input. We invert this assumption: specificity is the primary independent variable, and we measure how degradation propagates through the tool-calling pipeline as it decreases.

The closest methodological precedent is work on ambiguity in information-seeking dialogue. Aliannejadi et al. (2019) showed that one good clarifying question can improve retrieval precision by 170%+, showing how task success depends on resolving vagueness.

1.7. Thesis Organization

The remainder of this thesis is organized as follows.

Chapters 2 and 3 present the first case study: LAMINA, a tool-augmented system for materials science feasibility assessment. Chapter 2 describes the agent's architecture, tool registry, and routing design. Chapter 3 evaluates 37 claims across 3 models and 3 routing configurations, testing five pre-registered hypotheses about tool effectiveness, routing quality, ambiguity-driven instability, multi-call benefits, and model scale effects. The clearest finding is that linguistic ambiguity induces large, systematic instability in agent predictions (flip rate $= 0.43$), even when the underlying tools function correctly.

Chapters 4 and 5 present the second case study: PathFinder, a system for construct-

ing multi-step search strategies over VEuPathDB's pathogen genomics databases. Chapter 4 describes the strategy-construction task, system architecture, tool registry, and search-discovery mechanism. Chapter 5 evaluates 35 gold-standard strategies at three vagueness levels across two models (420 total runs). Underspecification collapses gene-set F_1 from 0.66 to 0.16, with the dominant degradation at the precise-to-moderate boundary where dataset-level specification is lost.

Chapter 6 synthesizes findings from both case studies, identifies common failure patterns across domains, derives design recommendations for scientific tool-calling agents, discusses shared limitations, and outlines future directions.

CHAPTER 2 : LAMINA: Agent Design for Materials Feasibility Assessment

2.1. Overview and Scientific Context

Materials science offers a favorable testbed for studying tool-augmented LLM agents. Unlike many scientific domains where computational tools are either proprietary or require bespoke simulation pipelines, materials informatics has a number of openly accessible, machine-callable toolchains, including:

- **CALPHAD (CALculation of PHAse Diagrams):** thermodynamic databases and equilibrium solvers, accessible via libraries such as `pycalphad` (Otis and Liu, 2017), that predict phase stability, solidification paths, and microstructure across composition-temperature space.
- **DFT databases:** curated first-principles data from the Materials Project (MP) (Jain et al., 2013), providing structural, energetic, elastic, and electronic properties for over 150,000 inorganic compounds.
- **Neural surrogates:** models such as CHGNet (Deng et al., 2023) that approximate DFT-level calculations at a fraction of the cost, enabling rapid screening of stability, diffusion barriers, and defect energetics.

For many materials science questions (*"Does this alloy form a brittle intermetallic at 600 K?"*, *"Is this phase a viable Li-ion anode?"*, ...) there exists both a classical path (query a thermodynamic database, run an equilibrium calculation) and an LLM path (reason from prior knowledge). The central question is how well an LLM can select, parameterize, and reconcile these tools when the input claim is expressed in natural language that may be vague, ambiguous, or underspecified.

DARPA's Scientific Feasibility (SciFy) program (Defense Advanced Research Projects Agency (DARPA), 2024) gives a formalization of this challenge: given a natural-language

technical claim, assess its feasibility by decomposing it into verifiable parts and checking those against trusted models, simulations, and data. Materials science is one of the core SciFy domains because the toolchains described above make many claims verifiable. So, the bottleneck shifts from "can we compute this?" to "can the agent correctly translate the claim into the right computation?"

This chapter presents LAMINA (**L**LM-**A**ssisted **M**aterial **IN**formatics and **A**nalysis), a tool-augmented system for materials informatics built on the Kani framework (Zhu et al., 2023). We describe the scientific motivation for the system, its architecture, tool subsystems, routing strategy, and design tradeoffs. The following chapter then evaluates LAMINA on the Drop-4 benchmark to study how tools, routing, model scale, and linguistic vagueness affect feasibility judgments.

2.2. Target Task and Domain Scope

LAMINA is designed to assess natural-language materials science claims by translating them into one or more tool calls, interpreting the returned evidence, and synthesizing it into a final feasibility judgment. Although the experimental chapter evaluates the system on the Drop-4 benchmark, the system itself is intended to cover a broader set of materials reasoning patterns that recur across materials informatics workflows.

2.2.1. Domain Coverage

Claims handled by LAMINA span five broad materials science areas:

- **Thermodynamics:** eutectic formation, phase stability, solidification behavior, and composition-temperature relationships.
- **Electrochemistry:** battery electrode voltages, anode viability, lithiation mechanisms, and capacity predictions.

- **Magnetism:** saturation magnetization, coercivity, doping effects on magnetic properties, and permanent magnet performance.
- **Defects and semiconductors:** dopant site preference, defect formation energies, and structural phase transitions.
- **Mechanics and alloys:** elastic properties, lattice parameter effects of solute additions, and alloy strengthening.

2.3. Design Goals

LAMINA is designed around four principles:

- **Reliable with domain tools:** The controller is built around thermodynamically and physically grounded tools (CALPHAD, DFT/Materials Project, CHGNet, etc.), with basic consistency checks (e.g., composition normalization, temperature ranges, phase constraints) to reduce obviously invalid calls and keep outputs aligned with established databases.
- **Transparent via streaming traces of tool calls:** Every tool call is exposed in an LLM- and human-readable trace: the system logs the tool name, arguments, return values, relevant external IDs (e.g., Materials Project IDs, TDB phase names), and any errors. These traces are emitted incrementally as server-sent events (SSE), enabling both interactive use and post-hoc replay.
- **Tool-rich and broad across materials workflows:** LAMINA exposes 50 domain-specific functions. The goal is to give the user a broad palette of physically grounded computations rather than a single calculator or database.
- **Reproducible:** We pin database and API versions (e.g., specific TDB files, Materials Project and CHGNet releases, and model names) and provide open-source code,

prompts, and configuration files so that other groups can re-run our experiments and extend the tool set.

2.4. Architecture and the Kani Framework

LAMINA's architecture follows a three-tier design:

1. **Frontend:** A React application that consumes a server-sent event (SSE) stream and renders a conversational interface with tool logs. The model's text appears as chat-style messages; each tool call and its trace can be inspected from a sidebar showing arguments, raw output, and metadata.

2. **Backend:** A FastAPI service that exposes SSE-streaming endpoints. Incoming requests contain a natural-language claim and metadata. The request is passed to a Kani-based (Zhu et al., 2023) controller that manages chat history, exposes the tool registry to the model, and executes model-generated function calls.

3. **Tool registry:** 50 domain-specific Python functions, each decorated with Kani's `@ai_function` decorator that provides schema hints (units, enums, valid ranges) to the LLM.

Kani. Kani (Zhu et al., 2023) is a lightweight, hackable framework for building tool-calling LLM applications. It wraps an LLM (in our case, OpenAI models) and a registry of typed Python functions, while handling the core tool-calling loop: exposing tools to the model, managing chat history and context, validating and invoking function calls, and returning tool outputs to the model for further reasoning. The model remains responsible for the reasoning itself, including whether to call a tool, which tool to call, and how to use the returned outputs. Figure 2.2 summarizes the division of responsibilities across the frontend, backend, and tool registry.

Figure 2.1: A sample query to generate a phase diagram using LAMINA. The sidebar shows the tool call trace, including arguments and return values.

Streaming and observability. Each interaction is emitted as a typed SSE event: model text, `tool_start` and `tool_end` events (with arguments, return values, latency, and external IDs such as Materials Project IDs), usage summaries, and optional images (e.g., phase diagrams, voltage curves). These traces serve two purposes: they drive the interactive frontend and they are logged for post-hoc analysis of failure modes.

2.5. Tooling Subsystems (50 Functions)

LAMINA exposes 50 domain tools grouped into eight subsystems (Table 2.1) and briefly described below. Each tool implements a narrow, typed operation (e.g., *"equilibrium phase fractions at a given* (x, T)*"* or *"bulk modulus of Ag from MP"*); The Kani-based controller exposes these tools to the model, which selects which tool to call based on the claim.

- **Thermodynamics (CALPHAD; 9 functions):** Wraps `pycalphad` (Otis and Liu, 2017) with assessed thermodynamic databases (COST 507 (Saunders and Miodownik,

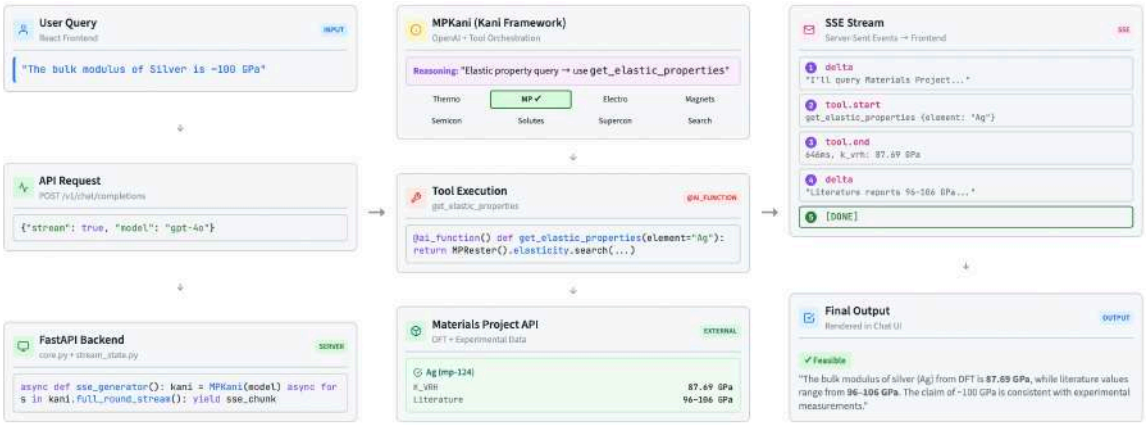


Figure 2.2: Backend architecture and tool orchestration. FastAPI hosts a Kani-based controller that selects tools, calls external services (CALPHAD, Materials Project, CHGNet, etc.), and emits SSE traces for each interaction.

1998) and MatCalc's `mc_al_v2.037.tdb` (Povoden-Karadeniz, 2025)). Core tools generate binary phase diagrams (`plot_binary_phase_diagram`), plot phase stability vs. temperature for fixed compositions (`plot_composition_temperature`), and compute equilibrium phase fractions at single points or over temperature ranges. Higher-level, domain-specific tools test natural-language assertions about phase formation across composition-temperature grids, including Scheil-Gulliver (Scheil, 1942; Gulliver, 1913) as-cast solidification behavior.

- **Materials Project (MP) / DFT / CHGNet (8 functions):** Expose MP-backed (Jain et al., 2013) structural, energetic, elastic, and electronic properties, with optional CHGNet surrogates (Deng et al., 2023). Basic search and retrieval by a chemical system, element, formula, or MP IDs; elastic property retrieval; alloy composition matching; and higher-level helpers for comparing material properties and analyzing doping effects.

- **Electrochemistry (8 functions):** Combine MP insertion-electrode models with convex-hull analysis. Tools find candidate battery electrodes for a given working ion, expose

Domain	Representative Functions	#
Thermodynamics	Equilibrium, phase fractions, Scheil	9
Materials Project/DFT/CHGNet	Stability, elastic, electronic	8
Electrochemistry	Voltage, capacity, insertion window	8
Magnetism/Defects/Alloys	M_s/T_c, site preference, diffusion, mechanics	11
Semiconductors	Octahedral distortion, defect site preference	8
Superconductors	Cuprate c-axis stability analysis	1
Solutes	Lattice parameter engineering	4
Search	Web and scientific literature search	1
Total		**50**

Table 2.1: Tool families in the LAMINA tool registry. Counts refer to distinct functions. A full function-level reference is provided in Appendix A.

voltage-capacity curves, check composition stability (optionally voltage-dependent), assess anode viability, and analyze lithiation mechanisms. One tool uses CHGNet NEB to estimate ion-hopping barriers in graphite/graphene galleries.

- **Magnetism / Defects / Alloys (11 functions):** Cover saturation magnetization, magnetic ordering, permanent magnet performance (B_r, H_c, $(BH)_{\text{max}}$, pull force), doping effects on M_s, and alloy microstructure/mechanics. Explicit guards ensure same-phase, same-space-group comparisons when evaluating doping effects. Alloy-focused tools connect CALPHAD equilibria to matrix/secondary phases and mechanical desirability.

- **Semiconductors (8 functions):** Focus on octahedral distortions, structural phase transitions, and point defects. Tools compute defect formation energies, analyze doping site preference, find doped variants with similar crystal symmetry, and compare undoped vs. doped magnetic properties.

- **Superconductors (1 function):** `analyze_cuprate_octahedral_stability` tar-

gets layered cuprates (e.g., La_2CuO_4, YBCO), relating c-axis spacing, Cu-O octahedral coordination, and family to trends in octahedral stability and Jahn-Teller distortions (Jahn and Teller, 1937).

- **Solutes (4 functions):** Analyze lattice-parameter effects and solubility in fcc matrices, combining CALPHAD solubility checks with Vegard's law (Vegard, 1921) and size-misfit theory (Liang et al., 2025).
- **Search (1 function):** A SearXNG-backed web search used as a fallback when claims reference systems outside the curated tool coverage.

2.6. Default Routing Policy and Guardrails

LAMINA's default operating mode uses a hand-engineered policy router that steers the model toward the appropriate tool family for a given claim. The goal is not to hard-code full reasoning trajectories, but to constrain tool selection enough that the controller reliably reaches the relevant scientific evidence before producing a verdict.

Policy router. The default Policy router organizes tools by claim type (thermodynamics, electrochemistry, magnetism, defects, semiconductors, and related domains) and exposes routing guidance in the system prompt. For example, thermodynamic claims are directed toward CALPHAD-backed equilibrium and phase-fraction tools, electrochemistry claims toward electrode and voltage-analysis tools, and doping or site-preference claims toward Materials Project, defect, or semiconductor tools as appropriate. This policy-based routing is the default configuration of LAMINA; later chapters evaluate it against weaker and stronger routing baselines.

Capability	LAMINA	HoneyComb	LLMatDesign	MatSciAgent	MatTools
Tool-using LLM agent	✓	✓	✓	✓	✓
DFT / Materials Project data	✓	–	✓	✓	✓
CALPHAD thermodynamic tools	✓	–	–	–	–
Neural surrogates (CHGNet)	✓	–	✓	–	–
Ordinal feasibility benchmark	✓	–	–	–	–
Ambiguity / flip rate analysis	✓	–	–	–	–
Streaming traces of tool calls	✓	–	–	–	–

Table 2.2: Comparison of LAMINA with prior materials science tool-augmented systems (Zhang et al., 2024; Jia et al., 2024; Chaudhari et al., 2025; Liu et al., 2025a).

Schema hints and argument constraints. Tools are described with schema hints through Kani's `@ai_function` decorator, including expected units, valid enums, and structured argument types. Kani also validates model-generated arguments against these schemas, correcting simple issues when possible and surfacing structured errors when it cannot. These hints help the model formulate tool calls in a way that is syntactically valid and semantically aligned with the underlying scientific APIs. In practice, they reduce common errors such as malformed compositions, invalid phase labels, or mis-specified numeric arguments.

Parameter guards. Beyond prompt-level routing, LAMINA applies lightweight guards to tool parameters before execution. These include composition normalization, temperature clamping to database validity ranges, phase constraints where applicable, and fallback defaults for missing arguments. The purpose of these guards is to reduce obviously invalid calls while preserving the controller's flexibility.

2.7. Positioning Relative to Prior Systems

Taken together, these design choices (domain-grounded tools, streaming observability, and structured routing) distinguish LAMINA from prior materials science agent systems. As

detailed in Chapter 1, LAMINA is the only such system combining CALPHAD tools, neural surrogates, ordinal feasibility evaluation, and ambiguity analysis (Table 2.2).

2.8. Chapter Summary

This chapter described LAMINA as a tool-augmented agent for materials feasibility assessment. We introduced the scientific setting that motivates the system, outlined its architecture and observability mechanisms, described its 50-function domain tool registry, and explained the routing and guardrail layer that connects natural-language claims to grounded scientific computations. The next chapter turns from system construction to empirical evaluation, focusing especially on how the agent behaves under prompt vagueness and underspecification.

CHAPTER 3 : LAMINA: Experiments on Vagueness and Underspecification

3.1. Benchmark, Claim Partition, and Research Questions

Chapter 2 described LAMINA as a tool-augmented agent for materials feasibility assessment. This chapter turns from system design to empirical evaluation. We evaluate LAMINA on Drop-4, a 37-claim benchmark from DARPA's Scientific Feasibility (SciFy) program, and study how tool use, routing quality, model scale, and linguistic vagueness affect the reliability of feasibility judgments.

3.1.1. The Drop-4 Benchmark

Drop-4 comprises 37 natural-language materials science feasibility claims from DARPA's SciFy program (Defense Advanced Research Projects Agency (DARPA), 2024), each labeled on a 5-point ordinal scale $\{-2, -1, 0, 1, 2\}$ by domain reviewers, where -2 indicates strong infeasibility and $+2$ indicates strong feasibility. The claims are written to resemble realistic materials-science statements: assertions about eutectic compositions, phase fractions as a function of temperature, Li-ion anode behavior, dopant site preference, changes in saturation magnetization under doping, and similar topics. Most can, in principle, be evaluated using one or more of the standard materials toolchains (CALPHAD/TDB, DFT databases, surrogates, or literature search).

3.1.2. Precise vs. Vague Claims

We partition Drop-4 into two subsets that are central to our analysis. This partition was created manually by inspecting each claim and grouping it according to whether it contained explicit numeric specifications or vague quantifiers requiring interpretation. Section A.5 in the appendix provides the full claim lists for both subsets.

- **Precise claims (15 of 37):** contain explicit numbers and comparators with no vague quantifiers. Example: *"The bulk modulus of silver is ∼100 GPa."*
- **Vague claims (22 of 37):** contain vague quantifiers, approximations, or underspecified comparators that require numeric or semantic interpretation before tool calls can be made. Examples include phrases like *"a few%"*, *"∼Al15Zn75"*, *"*$\approx 0.5\,V$*"*, or *"stronger magnet."*

Vague claims exhibit *ambiguous quantities and thresholds* (e.g., "a few%" $\rightarrow \{2, 3, 5\}$ at.%) and *ambiguous entities and references* (e.g., "stronger magnet" $\rightarrow$ higher M_s? higher coercivity? higher pull force?). This partition is used throughout the remainder of the chapter to analyze the effect of linguistic underspecification on agent behavior.

3.1.3. Research Questions

This chapter addresses five questions. First, do tools improve agreement with expert labels relative to no-tools baselines (H1)? Second, does routing quality matter, both in agreement and in avoiding missed tool families (H2)? Third, does linguistic vagueness induce instability, measured through flip rates, and lower agreement relative to precise claims (H3)? Fourth, do multiple tool calls improve performance over a single-call constraint (H4)? Finally, do tools reduce the performance gap between smaller and larger models (H5)?

3.2. Experimental Setup

3.2.1. Models

We compare three OpenAI models spanning a range of capability and cost:

- `gpt-4o-mini`: a small, efficient model;
- `gpt-4o`: a mid-range model;

- `gpt-5.1`: a high-reasoning configuration.

Each model is evaluated with and without tools. Model parameters are set to `temperature=0`, `top_p=1` across all tests to ensure reproducibility.

3.2.2. Configurations

Beyond the tools/no-tools comparisons, we evaluate two additional axes of variation: routing quality and tool-call multiplicity.

Routing conditions. We compare three routing configurations for `gpt-4o` with tools:

- **No-Policy Router:** A minimal system prompt with the task description only. All 50 tools are visible to the model, but no decision rules guide selection.
- **Policy Router (default):** The default LAMINA routing and guardrail layer described in Section 2.6. Hand-engineered decision rules steer the model toward appropriate tool families based on claim type, while schema hints and parameter guards constrain tool calls.
- **Oracle Router:** Manually specified and pre-called tools with a specific tool ordering and key parameters per claim. This serves as an upper bound on routing quality.

Tool-call multiplicity. We also compare a multi-call setting (LAMINA's default, where the model may invoke tools repeatedly) against a single-call constraint (exactly one tool invocation per claim), both using `gpt-4o` with the Policy router.

3.2.3. Evaluation Prompt

Each claim is presented to the model using a structured prompt template that frames the task and defines the ordinal scale:

```
Evaluate the following materials science claim and provide a feasibility
rating.

Claim: {claim_text}

Rate the claim's feasibility on a scale from -2 to +2:
-2: Strongly infeasible (contradicts known physics/data)
-1: Likely infeasible (evidence suggests unlikely)
0: Uncertain (insufficient evidence either way)
+1: Likely feasible (evidence suggests plausible)
+2: Strongly feasible (well-supported by data/theory)

Use any available tools to investigate the claim. Provide your reasoning,
then give your final verdict as: VERDICT: {integer}
```

This template is identical across all configurations (with-tools, no-tools, all routing modes). In the no-tools condition, the model receives the same prompt but has no tools available and must rely entirely on its prior knowledge.

3.2.4. Metrics

Quadratic Weighted κ (QWK). The primary metric is Quadratic Weighted κ (QWK) (Cohen, 1968), a special case of Cohen's weighted κ for ordered labels, using quadratic disagreement weights to penalize larger ordinal errors more heavily than exact-match accuracy while adjusting for label marginals. Given an $L \times L$ confusion matrix O between predicted and gold labels, with expected-by-chance matrix E, and a quadratic weight matrix W where $w_{ij} = (i-j)^2/(L-1)^2$:

$$\kappa_w = 1 - \frac{\sum_{i,j} w_{ij}\, O_{ij}}{\sum_{i,j} w_{ij}\, E_{ij}}$$

A κ_w of 1 indicates perfect agreement, 0 indicates chance-level agreement, and negative values indicate systematic disagreement. On the 5-point ordinal scale $\{-2, \ldots, +2\}$ used in Drop-4, a prediction that is off by one level incurs a cost of $1/16$, while a prediction off by four levels (e.g., -2 vs. $+2$) incurs a cost of 1.

Flip Rate (FR) and Quadratic-Weighted Flip Rate (QWFR). For the ambiguity analysis (H3), we expand each vague claim into M alternate numeric or semantic interpretations and re-run the system once per interpretation. The *flip rate* measures how often the predicted label changes across interpretations:

$$\mathrm{FR}(c) = \frac{2}{M(M-1)} \sum_{i<j} \mathbf{1}[\hat{y}_i \neq \hat{y}_j]$$

where $\hat{y}_i$ is the predicted label under interpretation i of claim c. Let $\mathcal{D}_A$ denote the set of vague claims. The dataset-level flip rate is

$$\mathrm{FR} = \frac{1}{|\mathcal{D}_A|} \sum_{c \in \mathcal{D}_A} \mathrm{FR}(c).$$

A high FR indicates that the system's output is unstable with respect to how a vague phrase is resolved.

The *quadratic-weighted flip rate* additionally weights each flip by its ordinal distance. Re-indexing labels from $\{-2, -1, 0, 1, 2\}$ to $\{0, \ldots, 4\}$ via $\tilde{y}_i = \hat{y}_i + 2$, we define

$$\mathrm{QWFR}(c) = \frac{2}{M(M-1)} \sum_{i<j} \left(\frac{|\tilde{y}_i - \tilde{y}_j|}{L-1} \right)^2$$

The dataset-level quadratic-weighted flip rate is

$$\mathrm{QWFR} = \frac{1}{|\mathcal{D}_A|} \sum_{c \in \mathcal{D}_A} \mathrm{QWFR}(c).$$

QWFR distinguishes between small flips (± 1 level) and large flips (e.g., $-2 \to +2$), which are more consequential for downstream decisions.

Secondary metrics.

- Exact-match accuracy on $\{-2, \ldots, 2\}$;
- Mean absolute error (MAE) on the ordinal scale;
- Number of tool calls;
- Tool miss-rate (fraction of claims where a required tool family is never invoked).

3.2.5. Statistical Analysis

Because Drop-4 is small ($N = 37$) and its label distribution yields a chance-level $\kappa \approx 0 \pm 0.165$ (95% interval $[-0.32, 0.33]$), raw κ estimates are noisy. For all pairwise comparisons, we:

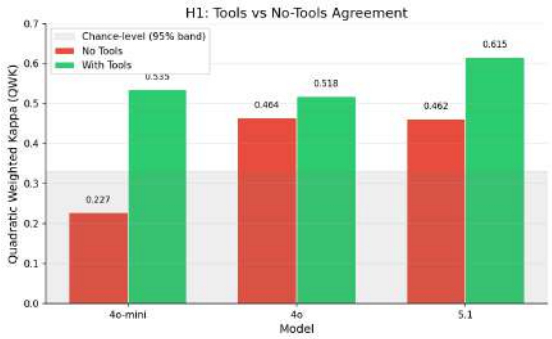
1. compute per-claim contributions to QWK,
2. estimate the distribution of $\Delta\kappa$ via non-parametric bootstrap with 10,000 resamples over claims, and
3. report $\widehat{\Delta\kappa}$ and its 95% bootstrap confidence interval.

We calibrate the following effect-size heuristic for this dataset: $|\Delta\kappa| < 0.20$ is indistinguishable from noise; $0.20 \leq |\Delta\kappa| < 0.35$ is borderline; and $|\Delta\kappa| \geq 0.35$ is very likely a real effect. A hypothesis is considered supported when the 95% CI has lower bound > 0 and $\widehat{\Delta\kappa} \geq 0.25$.

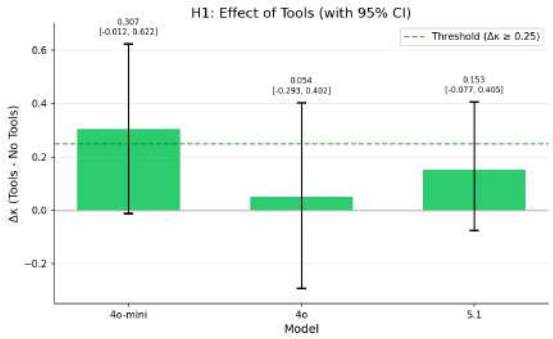
Reproducibility. All database, API, and model versions are recorded in Appendix A.

Model	Setting	QWK	$\Delta\kappa$	Acc	MAE
4o-mini	with-tools	0.535	+0.307	0.514	1.000
4o-mini	no-tools	0.227	—	0.243	1.405
4o	with-tools	0.518	+0.054	0.405	1.135
4o	no-tools	0.464	—	0.297	1.135
5.1	with-tools	0.615	+0.153	0.514	0.919
5.1	no-tools	0.462	—	0.432	1.162

Table 3.1: Agreement and efficiency across models and tool settings on Drop-4 ($N = 37$). $\Delta\kappa$ is computed relative to the corresponding no-tools configuration.



(a) QWK for each model with and without tools on Drop-4. The shaded band indicates the 95% chance-level region.



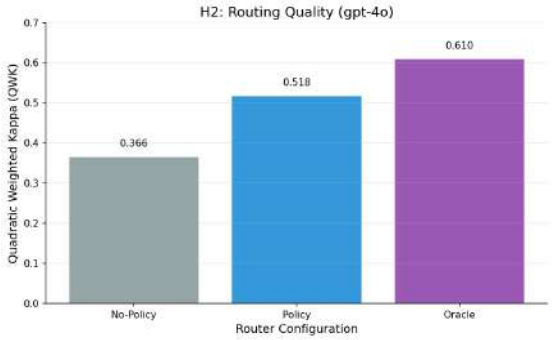
(b) Estimated effect of tools ($\Delta\kappa = \kappa_{\text{tools}} - \kappa_{\text{no-tools}}$) with 95% bootstrap CIs. The dashed line marks the preregistered threshold ($\Delta\kappa \geq 0.25$).

Figure 3.1: H1: Tools vs. no-tools on Drop-4.

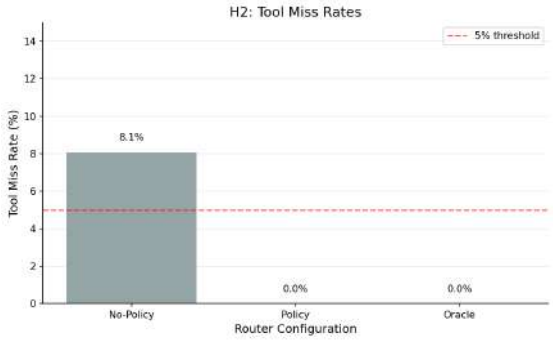
3.3. Results

3.3.1. Overall Agreement on Drop-4

All configurations achieve QWK values well above random chance (expected $\kappa \approx 0 \pm 0.165$). Across models and tool settings, QWK ranges from 0.227 to 0.615 on the 37 claims. Table 3.1 summarizes performance for each model and tool configuration. In each setting, `gpt-5.1` attains the highest QWK, with `gpt-4o` and `gpt-4o-mini` trailing.



(a) QWK for `gpt-4o` under three routing configurations: no-policy, LAMINA Policy router, and oracle router.



(b) Tool miss-rate for the same routing configurations. The dashed line indicates the preregistered 5% ceiling.

Figure 3.2: H2: Routing quality for `gpt-4o` on Drop-4.

3.3.2. H1: Tools vs. No-Tools

Figure 3.1a plots QWK for each model with and without tools; Figure 3.1b shows the corresponding $\Delta\kappa$ and 95% bootstrap confidence intervals.

For the smallest model, `gpt-4o-mini`, enabling tools increases QWK from 0.227 to 0.535 ($\widehat{\Delta\kappa} = 0.307$), nearly doubles accuracy ($0.243 \rightarrow 0.514$), and reduces MAE from 1.405 to 1.000. The 95% bootstrap CI for $\Delta\kappa$ is $[-0.012, 0.622]$. The point estimate is in the borderline range ($|\Delta\kappa| \approx 0.3$), but the CI slightly overlaps zero, so H1a is not formally supported under our pre-registered criteria.

For `gpt-4o`, tools produce a smaller change: QWK from 0.464 to 0.518 ($\widehat{\Delta\kappa} = 0.054$, 95% CI $[-0.293, 0.402]$), with modest accuracy gains ($0.297 \rightarrow 0.405$) and unchanged MAE. For `gpt-5.1`, tools increase QWK from 0.462 to 0.615 ($\widehat{\Delta\kappa} = 0.153$, 95% CI $[-0.077, 0.405]$) and reduce MAE from 1.162 to 0.919.

Because the 95% CIs include zero for all three models, H1a and H1b are not statistically supported, although point estimates consistently favor the with-tools setting.

3.3.3. H2: Routing Quality

We compare three routing configurations for `gpt-4o` with tools:

- **No-Policy:** QWK $= 0.366$, accuracy $= 0.351$, miss-rate $= 0.081$.
- **Policy (default):** QWK $= 0.518$, accuracy $= 0.405$, miss-rate $= 0.000$.
- **Oracle:** QWK $= 0.610$, accuracy $= 0.378$, miss-rate $= 0.000$.

H2a (Policy vs. No-Policy). $\widehat{\Delta\kappa}_{\text{Policy}} = 0.151$ (95% CI $[-0.085, 0.419]$) as shown in Figure 3.2a. The magnitude is borderline, but the CI overlaps zero. H2a is not supported.

H2b (Miss-rate). The Policy router achieves a miss-rate of 0, with the 95% CI upper bound below 5% as shown in Figure 3.2b. H2b is supported: the router reliably avoids missing manually required tool families.

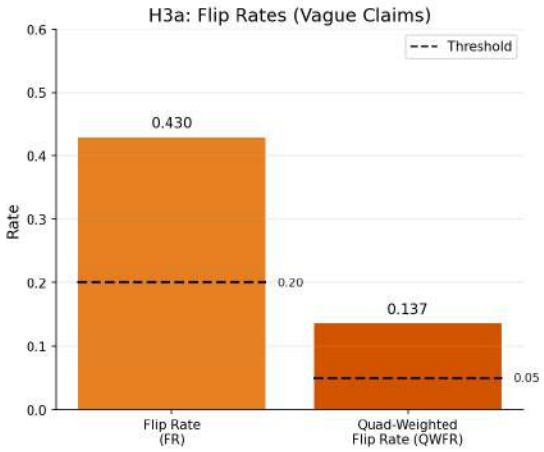
H2c (Oracle headroom). $\widehat{\Delta\kappa}_{\text{Oracle}} = 0.092$ (95% CI $[-0.021, 0.224]$). The CI overlaps zero, suggesting limited headroom from perfect routing alone. H2c is not supported.

Overall, routing appears helpful operationally (zero miss-rate) and plausibly boosts agreement, but the effect on QWK is not large enough on Drop-4 to satisfy pre-registered thresholds.

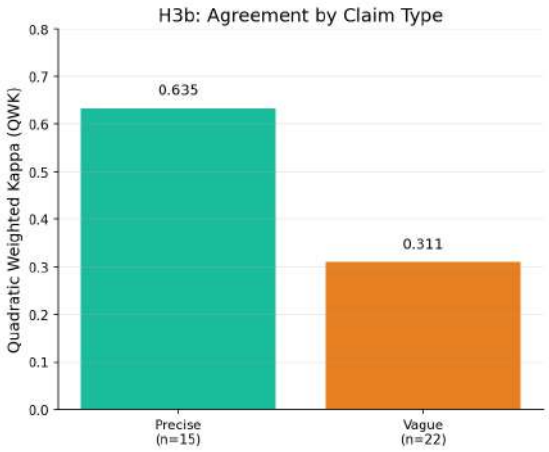
3.3.4. H3: Ambiguity, Flip Rates, and Agreement

Using `gpt-4o` with tools and the Policy router:

- Precise subset P (15 claims): $\kappa(P) = 0.635$
- Vague subset V (22 claims): $\kappa(V) = 0.311$



(a) Flip rate (FR) and quadratic-weighted flip rate (QWFR) on vague claims under alternate numeric mappings.



(b) QWK on precise vs. vague claims for `gpt-4o` with tools and the Policy router.

Figure 3.3: H3: Flip rates and agreement metrics on Drop-4.

The point-estimate gap is $\widehat{\Delta\kappa} = 0.323$, but the 95% bootstrap CI is wide: $[-0.337, 0.826]$ (Figure 3.3b). H3b (ambiguity lowers QWK) is not formally supported due to limited statistical power.

For H3a, we expand each vague claim into multiple numeric or semantic interpretations (e.g., "a few%" $\to \{2, 3, 5\}$ at.%) and re-run the system once per mapping. Over V:

$$\text{FR}(V) = 0.430 \quad (95\% \text{ CI } [0.271, 0.589]) \tag{3.1}$$

$$\text{QWFR}(V) = 0.137 \quad (95\% \text{ CI } [0.060, 0.227]) \tag{3.2}$$

Both are significantly above zero with 95% CIs entirely > 0, meeting the pre-registered thresholds of FR ≥ 0.20 and QWFR ≥ 0.05 (Figure 3.3a). H3a is supported: ambiguity induces substantial label flips, and these flips are often large in ordinal distance.

For the vague subset, the best and worst numeric mappings yield $\kappa_{\text{best}} \approx 0.790$ and $\kappa_{\text{worst}} \approx 0.098$: a range of nearly 0.7 in QWK driven entirely by how vague quantifiers are interpreted.

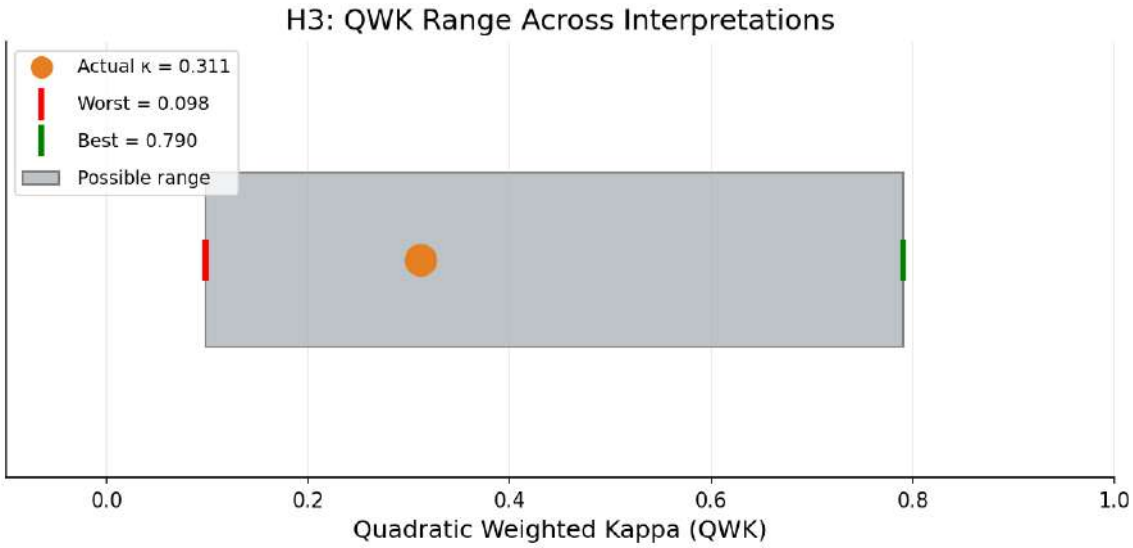


Figure 3.4: Range of possible QWK on vague claims for best vs. worst numeric remappings of ambiguous quantifiers.

3.3.5. H4: Multiple Tool Calls vs. a Single Call

Comparing multi-call (default) to single-call for `gpt-4o`:

- Multi-call: $\kappa = 0.518$, accuracy $= 0.405$, MAE $= 1.135$
- Single-call: $\kappa = 0.454$, accuracy $= 0.351$, MAE $= 1.243$

$\widehat{\Delta\kappa}_{\text{multi}} = 0.064$ (95% CI $[-0.128, 0.282]$). H4 is not supported: a single well-chosen tool call captures most of the achievable signal on Drop-4.

3.3.6. H5: Model Size and Diminishing Returns

Without tools:

$$\text{Gap}_{\text{no-tools}} = \kappa_{5.1} - \kappa_{4o\text{-mini}} = 0.462 - 0.227 = 0.235 \quad (95\%\ \text{CI}\ [-0.094, 0.563])$$

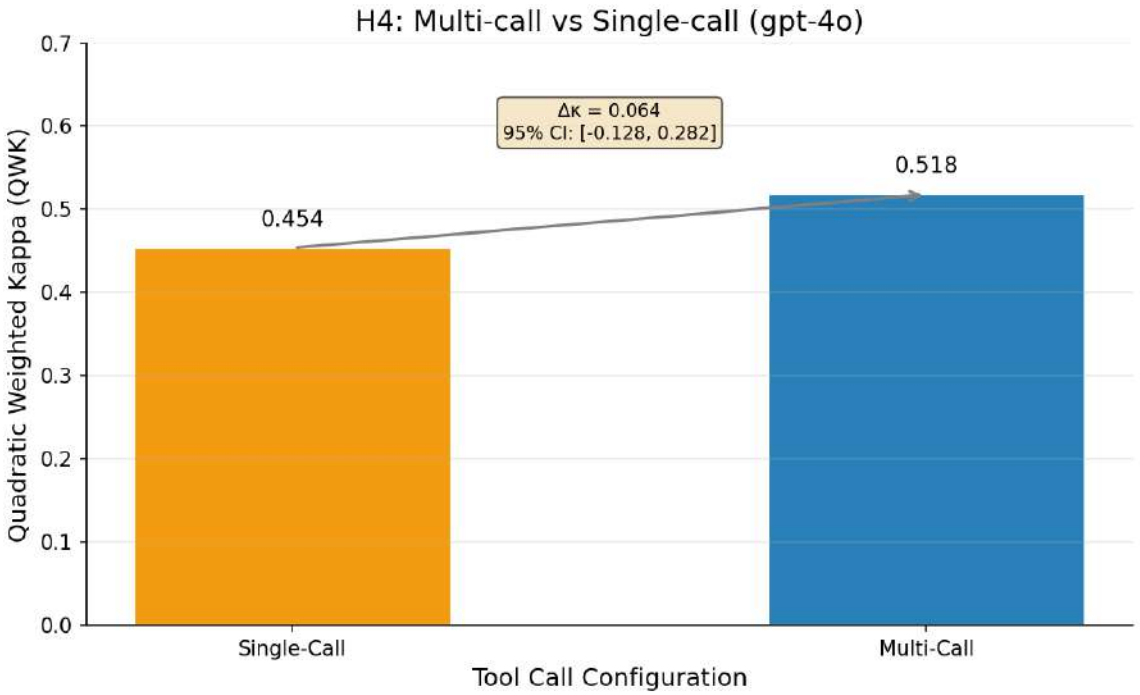


Figure 3.5: Agreement (QWK) for `gpt-4o` with tools under single-call vs. multi-call settings. Arrow annotation shows $\Delta\kappa$ and its 95% confidence interval.

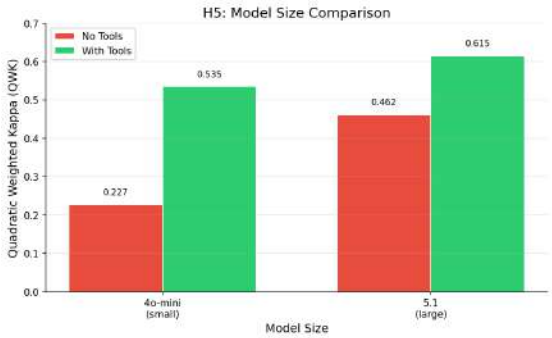
With tools:

$$\text{Gap}_{\text{tools}} = 0.615 - 0.535 = 0.081 \quad (95\%\ \text{CI}\ [-0.128, 0.320])$$

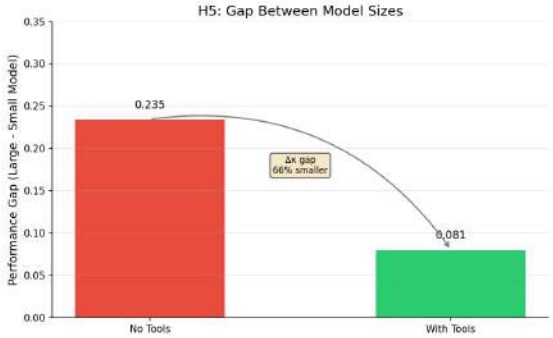
The gap shrinks from ≈ 0.24 to ≈ 0.08 (a reduction of roughly two thirds), and with tools the gap is small and statistically compatible with zero, satisfying the qualitative part of H5b. However, the shrinkage itself ($\Delta_{\text{gap}} = 0.154$, 95% CI $[-0.170, 0.478]$) has a CI that includes zero. H5a, H5b (qualitatively), and H5c are not statistically supported at this sample size.

3.3.7. Summary of Hypothesis Verdicts

Table 3.2 summarizes the outcome of each pre-registered hypothesis.



(a) QWK for small (`gpt-4o-mini`) vs. large (`gpt-5.1`) models with and without tools.



(b) Gap in QWK between `gpt-5.1` and `gpt-4o-mini` with and without tools. Annotation reports the relative reduction in the gap.

Figure 3.6: H5: Model size and diminishing returns with tools on Drop-4.

Hypothesis	**Verdict**	**Key evidence**
H1a/b: Tools improve QWK	Not supported	CIs overlap 0
H2a: Policy routing helps QWK	Not supported	CI overlaps 0
H2b: Miss-rate $< 5\%$	Supported	0% miss-rate
H2c: Oracle headroom	Not supported	CI overlaps 0
H3a: Ambiguity causes flips	Supported	FR $= 0.43$, QWFR $= 0.14$
H3b: Ambiguity lowers QWK	Not supported	Wide CI
H4: Multi-call helps	Not supported	Small effect
H5a/b/c: Tools compress gap	Not supported	CIs overlap 0

Table 3.2: Summary of pre-registered hypothesis verdicts on Drop-4. Only H2b (routing eliminates tool misses) and H3a (ambiguity induces flip rate instability) are formally supported.

3.4. Failure Modes and Underspecification Effects

3.4.1. Why Most Hypotheses Are Not Supported

The results in Section 3.3 present an apparent paradox: point estimates consistently favor tools, routing, and larger models, yet most hypotheses are "not supported." The explanation is purely statistical: Drop-4 has only 37 claims. Even when absolute QWK values are clearly above chance, we need differences of $\Delta\kappa \approx 0.25$–$0.35$ to obtain non-overlapping 95% confidence intervals on the same 37 claims. The dataset is large enough to reject pure chance for individual configurations but too small to firmly accept or reject most comparative hypotheses.

This is an important methodological lesson: small, high-quality benchmarks are valuable for studying phenomena (such as ambiguity-driven instability) but insufficient for resolving small-to-moderate effect sizes between system configurations.

3.4.2. Ambiguity as the Dominant Signal

The clearest supported finding is H3a: linguistic ambiguity induces large, systematic label flips. When vague claims are expanded into multiple plausible numeric or semantic interpretations, the system's predictions shift substantially, often across multiple ordinal levels (QWFR $= 0.137$).

The mechanism is consistent across examples: the tools behave correctly once given precise inputs, but the mapping from natural language to those inputs is unstable. Systems are internally consistent conditional on a precise interpretation, while being highly unstable with respect to the linguistic choices in the original claim.

3.4.3. Illustrative Examples

Example 1: Numeric ambiguity (Al–Zn eutectic). Consider the claim: *"Al and Zn form an eutectic at ∼Al15Zn75."* We instantiate "∼Al15Zn75" via three composition windows: tight ($\pm 1\%$: Al16Zn84-Al17Zn83), moderate ($\pm 5\%$: Al11Zn89-Al22Zn78), and loose ($\pm 10\%$: Al6Zn94-Al27Zn73).

Under the tight interpretation, the system predicts -2 (strongly infeasible); under moderate and loose interpretations, it predicts $+2$ (strongly feasible). The gold label is $+2$. The underlying CALPHAD calculations are correct in each case; what changes is which composition window the "$\sim$" symbol maps to.

Example 2: Semantic ambiguity (Fe_2O_3 "stronger magnet"). Consider the claim: *"Someone made a stronger magnet by doping α iron oxide (Fe_2O_3 trigonal) with aluminum."*

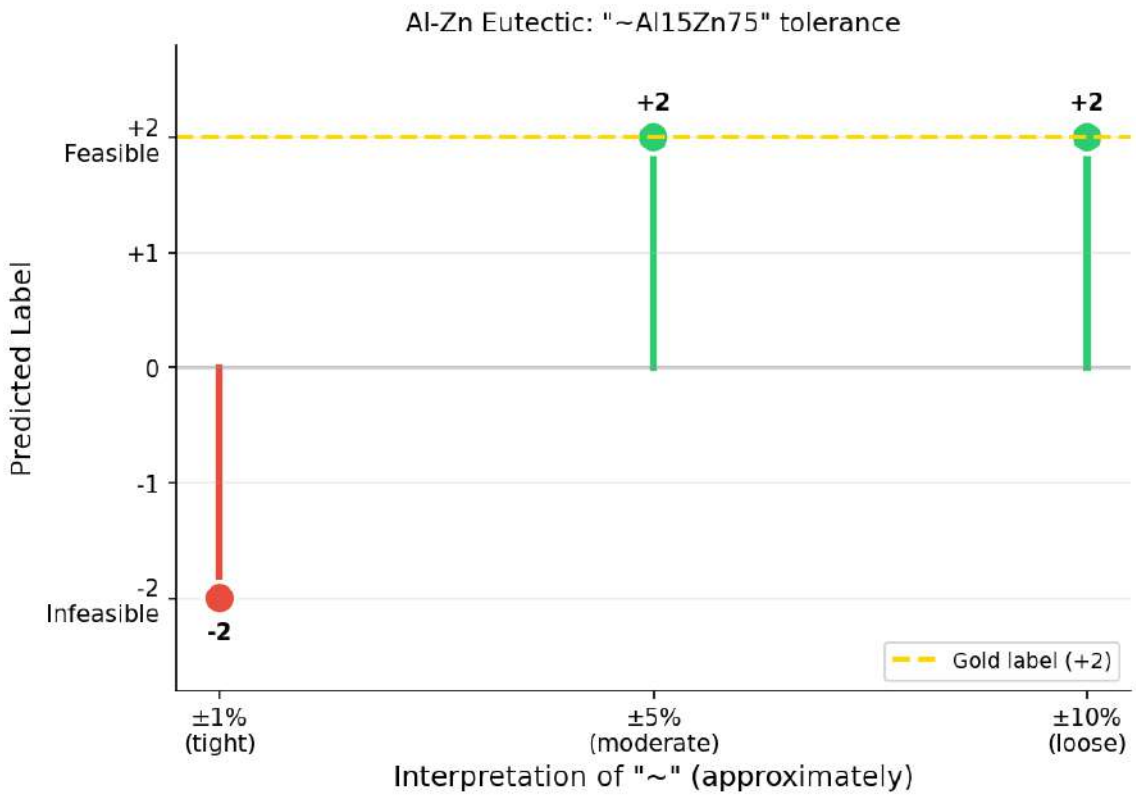


Figure 3.7: Ambiguity-induced flips for the Al-Zn eutectic claim (*"Al and Zn form an eutectic at ∼Al15Zn75"*). Three interpretations of "$\sim$" (a tight $\pm 1\%$ window, a moderate $\pm 5\%$ window, and a loose $\pm 10\%$ window around Al15Zn75) yield predicted labels of -2, $+2$, and $+2$, respectively, for a gold label of $+2$.

The ambiguity here is semantic: what does "stronger" mean? We encode four definitions: (i) higher saturation magnetization M_s, (ii) higher coercivity, (iii) higher pull force, and (iv) higher remanence. These yield predicted labels of -1, $+1$, -2, and -2, respectively, for a gold label of $+2$.

Again, the tools perform the same underlying calculation in each case; what varies is which magnetic descriptor is treated as the target of improvement.

Example 3: Parameterization crossing a decision boundary (Al-Mg-Zn τ-phase). The claim involves "a few percent" Zn in an Al–Mg matrix and asks whether the τ-phase forms. The three mappings of "a few%" yield strikingly different CALPHAD results:

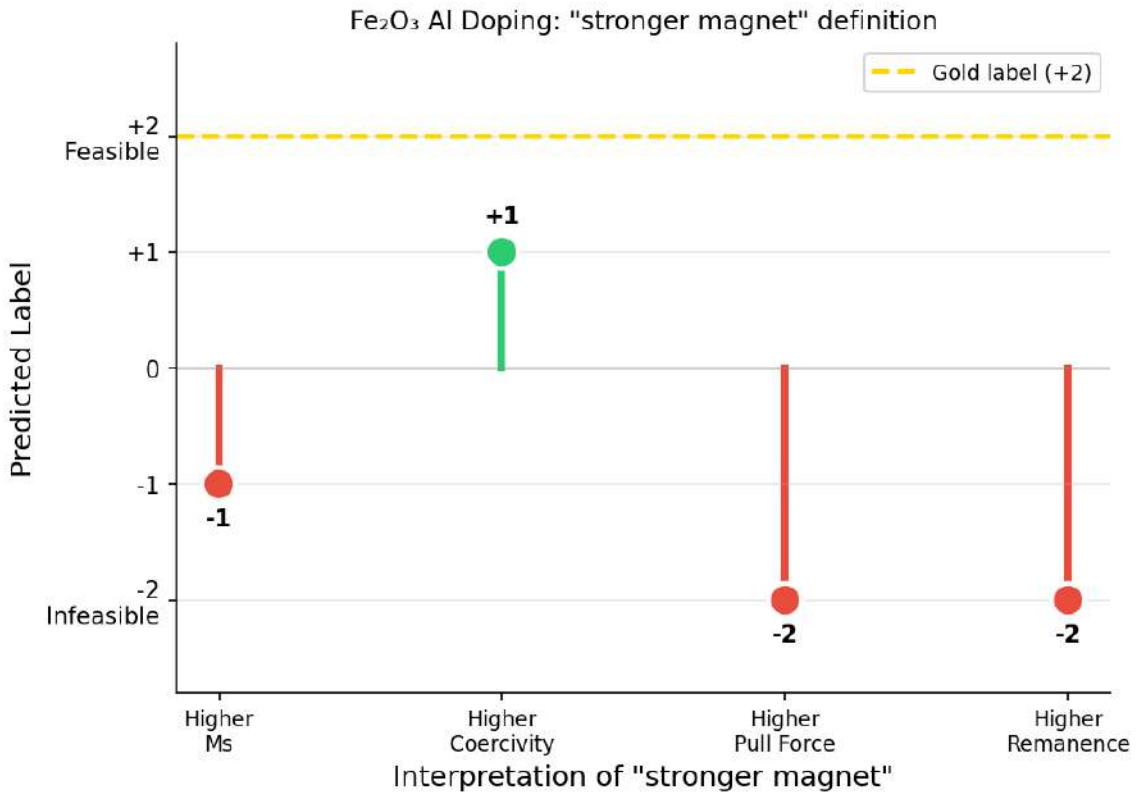


Figure 3.8: Ambiguity-induced flips for the Fe_2O_3 Al-doping claim ("Someone made a stronger magnet by doping α-Fe_2O_3 with Al"). Interpreting "stronger magnet" as higher M_s, higher coercivity, higher pull force, or higher remanence yields predicted labels of -1, $+1$, -2, and -2, respectively, for a gold label of $+2$.

- 2% Zn $\rightarrow$ τ-phase fraction $= 12.1\%$ $\rightarrow$ predicted -2 (infeasible)
- 3% Zn $\rightarrow$ τ-phase fraction $= 21.9\%$ $\rightarrow$ predicted $+2$ (feasible)
- 5% Zn $\rightarrow$ τ-phase fraction $= 41.5\%$ $\rightarrow$ predicted $+2$ (feasible)

The CALPHAD equilibrium calculation is correct in every case. The instability arises because "a few percent" maps to compositions that straddle a phase-formation boundary: at 2% Zn the τ fraction is below the threshold the model considers significant, while at 3% and above it crosses that threshold. This is the most concrete illustration of how a seemingly minor linguistic choice can cross a physical decision boundary and flip the verdict by four ordinal levels.

In practice, domain specialists may have a shared understanding of what "a few percent" means in a given metallurgical context: the phrase carries implicit precision grounded in years of training and familiarity with typical composition ranges.

However, if tool-augmented LLMs are to be integrated into scientific workflows, we cannot assume they bring this same tacit knowledge. Without explicit disambiguation, the model must map vague language to numeric tool inputs using only its training distribution, which may not reflect the conventions of a specific subfield. This is not a failure of the tools themselves, but a failure of the natural-language interface to constrain the space of reasonable interpretations before the tools are invoked.

3.4.4. Taxonomy of Failure Modes

Viewed through the lens of these results, the main failure modes are:

1. **Selection errors:** The router picks an inappropriate tool family or neglects a necessary one (e.g., answering a battery-voltage claim using only Materials Project band gaps). The Policy router eliminates manual miss-cases (H2b), but selection errors still appear when claims could reasonably invoke multiple tools.

2. **Parameterization errors:** Ambiguous phrases such as "a few%", "about 50 at.%", or "low temperature" are mapped to numerical values that are inconsistent across runs or misaligned with domain expectations. The CALPHAD reasoning is coherent once the numbers are fixed, but the mapping from language to numbers is not.

3. **Verbalization drift:** The final natural-language explanation sometimes deviates from the tool outputs: misreading phase fractions, mis-stating voltage plateaus, or reintroducing hedging that effectively changes the predicted label. For example, on a claim about an Fe-Al ferroalloy at 65 at.% Al, the CALPHAD tool correctly returns Al_2Fe

at 94.4% and `BCC_A2` at 5.6%, but the model's final explanation focuses on the minority `BCC` phase and characterizes it as a *"small amount of disordered solid solution,"* effectively understating the dominant intermetallic and shifting the verdict.

3.5. Discussion and Takeaways

3.5.1. What We Learned

Tools help, but the effect is hard to prove at small scale. All three models show higher QWK with tools than without, and the smallest model (`gpt-4o-mini`) shows the largest gain ($\Delta\kappa \approx 0.31$). But on 37 claims, these gains are statistically fragile. Future work related to LAMINA, and as demonstrated in PathFinder, should evaluate on larger, multi-domain benchmarks to resolve these effects.

Routing matters operationally. The Policy router eliminates tool miss-cases (H2b) and plausibly improves QWK, even though the improvement is not statistically decisive. This suggests that practical deployment should prioritize reliable tool selection: missing the right tool family entirely is a more consequential failure than sub-optimal parameterization. This kind of prompt-level routing is most plausible in closed-domain settings with a stable tool registry and a well-characterized task distribution; broader open-domain deployments would likely require learned routing, interactive clarification, or stronger external grounding.

Underspecification is the dominant source of instability under our evaluation metrics. H3a is the clearest and most robust finding: vague quantifiers and underspecified comparators induce a mean flip rate of 0.43 and a QWFR of 0.14, both with confidence intervals entirely above zero. The range of possible QWK on the vague subset spans nearly 0.7 depending on the model's interpretation.

3.5.2. LAMINA-Specific Limitations

- **Benchmark coupling:** LAMINA's Policy router and tool subsystems were developed alongside Drop-4, raising the possibility that routing rules and tool implementations are overfit to the 37 claims in the benchmark rather than generalizable to arbitrary materials science queries.
- **Label uncertainty:** Some vague claims have inherently uncertain gold labels, as experts might disagree on the correct feasibility rating.

Limitations shared with PathFinder are discussed in Chapter 6.

3.5.3. Connection to PathFinder

The LAMINA case study establishes two points that motivate the second case study. First, among the factors investigated here (tool use, routing quality, model scale, and prompt specificity) linguistic underspecification is the clearest source of instability. When a materials-science claim contains vague quantities or underspecified comparators, small changes in interpretation can produce different tool inputs and different final feasibility judgments.

Second, LAMINA studies this phenomenon in a relatively simple output setting: the agent produces a single ordinal label. This leaves open a broader question: what happens when the output is not a scalar judgment but a structured, executable artifact whose correctness depends on multiple sequential decisions? In the terminology of Chapter 1, this shift makes composition especially important: correctness now depends not only on choosing the right tools and parameters, but also on chaining intermediate outputs across multiple steps without error.

PathFinder addresses that question in a different scientific domain. Instead of mapping a natural-language claim to one or more scientific tool calls and a final label, it maps a natural-language biological question to a multi-step search strategy over VEuPathDB. This

makes the construction problem more fragile: the agent must choose the right searches, set their parameters correctly, and combine them with the right Boolean operators. PathFinder therefore tests whether the same underspecification effects observed in LAMINA persist (and possibly intensify) when the task requires building a complex structured object rather than making a single judgment.

3.6. Chapter Summary

This chapter evaluated LAMINA on Drop-4 under a range of model, tool, and routing configurations. Although most pairwise differences in QWK were not statistically decisive at this sample size, the results show consistent directional benefits from tools and routing, and they identify linguistic vagueness as the clearest source of instability. Across the ambiguity experiments, small changes in how vague language is resolved can produce large changes in tool inputs and final ordinal judgments.

CHAPTER 4 : PathFinder: Agent Design for VEuPathDB Strategy Construction

4.1. Overview and Scientific Context

Chapters 2 and 3 studied LAMINA, a tool-augmented agent for materials science feasibility assessment. In that setting, the task was to evaluate a single scientific claim and produce an ordinal feasibility judgment. The main finding was that, among the factors investigated, linguistic underspecification was the clearest source of instability: once a claim contained ambiguous quantities or underspecified comparators, small changes in interpretation could produce different tool inputs and different final verdicts. PathFinder asks the same broad question in a different scientific setting. Instead of producing a single label, the agent must construct a multi-step search strategy over a pathogen genomics database, so errors can arise not only in interpretation but also in search selection, parameterization, and Boolean composition.

Eukaryotic pathogen genomics provides a natural testbed for this second case study. Genomics research often requires constructing multi-step search strategies: structured query trees that combine heterogeneous data sources such as gene expression profiles, protein domain annotations, ortholog relationships, and single nucleotide polymorphism (SNP) variation into a single result set of candidate genes.

VEuPathDB (the Eukaryotic Pathogen, Vector, and Host Informatics Resources) is a consortium of 14 databases that collectively span pathogens responsible for malaria (*Plasmodium*), toxoplasmosis (*Toxoplasma*), trypanosomiasis (*Trypanosoma*), fungal infections (such as *Candida*, *Aspergillus*, and *Cryptococcus*) along with their arthropod vectors (such as *Anopheles*), and more. Thirteen of those databases support omics data mining through the Strategies Web Development Kit (WDK), a search system in which users compose multi-step query trees by combining individual searches with Boolean operators (intersect, union, minus, …) and unary transforms such as ortholog mapping across species (Amos et al.,

2022; Fischer et al., 2011).

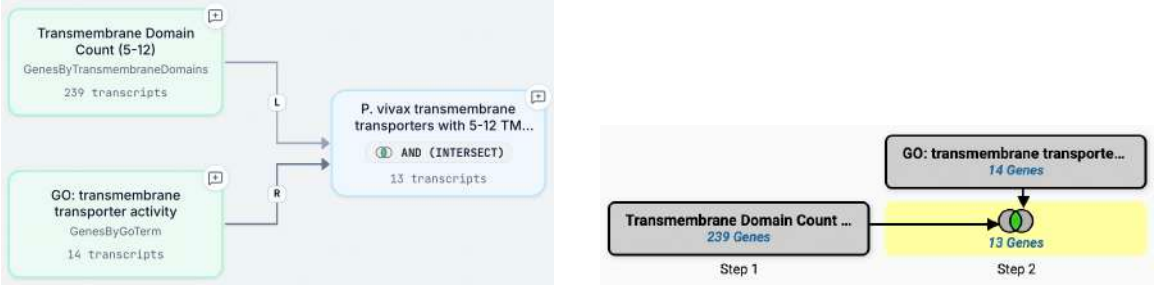
The central challenge in this domain is not only choosing the right data source, but also assembling an entire executable search strategy from a natural-language biological question. Researchers currently build these strategies interactively in a graphical interface, iteratively refining them until the output gene set matches their intent. The process is powerful, but it requires deep familiarity with the site's search vocabulary, parameter schemas, and dataset inventory. This creates an expertise mismatch: the researchers who most need these searches are often experts in pathogen biology, genomics, or infectious disease rather than in database internals, search schemas, or API behavior. Many researchers do eventually learn to use the interface effectively, but the barrier to entry is high.

This chapter presents PathFinder, a tool-augmented LLM system designed to construct these multi-step strategies from natural-language descriptions. The focus here is the design of the system itself: the task abstraction, architecture, tool registry, search discovery mechanism, and design tradeoffs. The following chapter then evaluates PathFinder on a corpus of gold-standard strategies under varying levels of prompt vagueness to study how faithfully the agent reproduces expert search strategies.

4.2. Target Task and Strategy Abstraction

The central abstraction in PathFinder is a **strategy**: a recursive tree of search steps where each internal node combines one or two inputs. Each leaf is a parameterized search (e.g., *"genes annotated with Gene Ontology (GO) term GO:0016301 in* P. falciparum*"*), and each internal node is a Boolean combination or transform. In practice, researchers build strategies in one of the VEuPathDB interfaces by adding searches, testing result counts, and refining parameters until the resulting gene set matches their biological question.

PathFinder is designed to automate this construction process. Given a natural-language



(a) PathFinder's graph editor view.

(b) PlasmoDB's native strategy workspace.

Figure 4.1: A concrete example strategy for finding likely membrane transporters in *P. vivax* P01. Two leaf searches are combined with an intersect operator: one filters genes by transmembrane domain count (5–12 domains), and the other filters genes by the GO term *transmembrane transporter activity* (GO:0022857). The final result contains 13 genes.

description of a biological goal, the agent must discover appropriate WDK searches, determine their parameters, assemble them into a valid recursive tree, and execute that tree against the live database. The result is a structured, executable strategy whose correctness depends on selecting the right searches, setting their parameters appropriately, combining them with the right operators, and producing the intended final gene set.

For each search, parameter refinement compounds the difficulty: researchers often need to calibrate thresholds against known positive and negative control gene sets, a trial-and-error process that requires multiple round-trips through the interface. PathFinder therefore treats strategy construction as a stateful search-and-build problem rather than a one-shot text generation task.

4.2.1. A Concrete Example of a Search Strategy

A strategy is easier to understand through a concrete example than through an abstract definition. Figure 4.1 shows a small strategy for finding *Plasmodium vivax* genes that are likely to be membrane transporters.

This example illustrates the four main pieces of a strategy. First, a *leaf search* is a single biological filter. Here, one leaf search keeps genes with 5–12 predicted transmem-

Figure 4.2: PathFinder's interface during strategy construction for the `pvivax_tm_transporter` strategy. The conversation panel shows the agent's reasoning and proposed strategy, while the bottom bar displays the minimized strategy graph: a `GenesByGoTerm` leaf node and a `GenesByTransmembraneDomains` leaf node joined by an intersection operator.

brane domains, and the other keeps genes annotated with the GO term for transmembrane transporter activity. Second, a *parameter* is the exact input given to a search, such as the organism name, the minimum and maximum number of transmembrane domains, or the GO term identifier. Third, a *Boolean operator* determines how two search results are combined. In this case, intersect keeps only genes that satisfy both conditions. Fourth, the *result count* records how many genes remain after the strategy has been applied; here the final result is 13 genes.

The rest of this chapter describes how PathFinder constructs strategies like this automatically from natural-language prompts. Figure 4.2 shows what that construction process looks like during an actual interaction.

4.3. Design Goals

PathFinder is designed around four principles:

- **Grounded in the WDK search vocabulary:** All strategy construction is mediated through VEuPathDB's REST API. Search names, parameter schemas, and result counts come from the live platform. This ensures that the agent can only propose searches that actually exist and parameters that are actually valid.
- **Transparent via streaming traces:** Every tool call is emitted as a server-sent event (SSE). For PathFinder, this includes search discovery queries, parameter lookups, step creation calls, and strategy build results. The full conversation, including intermediate reasoning and tool call arguments, is logged for post-hoc analysis.
- **Incremental and recoverable:** Strategy construction is incremental: each step is added to a local graph, validated, and pushed to the WDK individually. If a step fails validation, the agent can inspect the error and adjust without losing prior work. An undo mechanism preserves history snapshots after each mutation.
- **Reproducible:** Each evaluation run is fully logged, recording the model, conversation transcript, tool calls, and timing, enabling independent verification and re-analysis without re-running the agent.

4.4. Architecture and the Kani Framework

PathFinder uses a five-layer architecture (Figure 4.3) to support stateful, multi-step strategy construction and future integration into VEuPathDB:

1. **AI Agent Layer:** A `PathfinderAgent` class extending Kani (Zhu et al., 2023) that serves as the unified reasoning core. PathFinder composes tools from multiple mixin

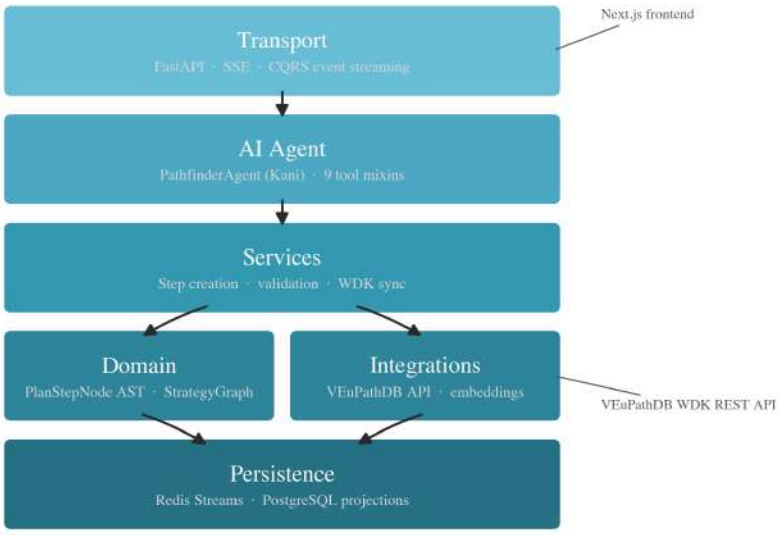


Figure 4.3: PathFinder's five-layer architecture.

classes into a single agent instance: catalog discovery, strategy graph mutation, step execution, result retrieval, conversation management, research (web and literature search), gene resolution, parameter optimization, and experiment validation. The agent sees all tools simultaneously and decides implicitly whether to plan (call discovery tools) or execute (call strategy mutation tools).

2. **Domain Layer:** Strategies are represented as recursive trees of `PlanStepNode` objects, where each node has zero (leaf search), one (transform), or two (Boolean combine) inputs. The node's *kind* (search, transform, or combine) is inferred structurally from the presence or absence of inputs and operators, rather than stored as an explicit field. A `StrategyGraph` session object maintains the working state: all steps, their WDK step IDs, result counts, and undo history. This state is not left implicit in prior tool calls alone: the agent can inspect the current strategy explicitly through inspection tools such as `get_strategy`, rather than reconstructing it only from con-

versation history.

3. **Service Layer:** Business logic for step creation, parameter validation, and WDK synchronization. Step creation is a pipeline: resolve inputs, validate the record type, check the search name against the catalog, canonicalize parameters, build the AST node, and push to the WDK. Validation uses a callback-based design: the service layer accepts `ValidationCallbacks` (async functions for record-type resolution, vocabulary lookup, and error formatting) rather than performing I/O directly, enabling testing without mocks.

4. **Integration Layer:** Adapters for external systems: the VEuPathDB REST API (search catalog, step CRUD, strategy management) and semantic search discovery. All integrations are accessed through protocol-based interfaces (`StrategySyncAPI`, `StepCountAPI`).

5. **Transport and Persistence:** FastAPI HTTP endpoints with SSE streaming. The event system follows a CQRS pattern: events are appended to Redis Streams for real-time consumption by the frontend, then projected into PostgreSQL for durable, read-optimized history.

4.5. Tool Registry

PathFinder exposes 44 tools to the agent across ten functional categories (Table B.1). The core strategy-construction subset most relevant to the system design consists of 19 tools across five categories (Table 4.1). These cover the tasks of search discovery, strategy mutation, execution, inspection, and external research. Additional utilities for export, conversation management, workbench analysis, optimization, experiment validation, step attachments, and planning artifacts are part of the broader registry but omitted here for brevity.

Category	Representative tools	#
Catalog discovery	`search_for_searches`, `get_search_parameters`	9
Strategy mutation	`create_step`, `update_step`, `delete_step`, `undo_last_change`	5
Execution & results	`get_estimated_size`, `get_sample_records`	2
Inspection	`get_strategy`	1
Research	`web_search`, `literature_search`	2

Table 4.1: Core strategy-construction tools exposed by PathFinder, grouped by category. These 19 tools cover discovery, graph mutation, execution, inspection, and external research. Additional utilities are part of the broader 44-tool registry but omitted here for brevity.

4.6. Search Discovery and Semantic Retrieval

A core challenge in strategy construction is *search selection*: given a natural-language description of what the researcher wants (e.g., "genes upregulated in gametocyte stages"), which of the hundreds of WDK searches should the agent use? PathFinder addresses this through semantic search over a precomputed index of search metadata.

Each WDK search is embedded using its name, display name, summary, description, parameter names, and associated dataset metadata via a sentence-transformer model (`nomic-ai/nomic-embed-text-v1.5`). At query time, `search_for_searches` embeds the user's natural-language description, retrieves the top-k nearest neighbors by cosine similarity, and applies keyword boosting: if any user-provided keyword appears as a substring of a search name, that search's similarity score receives a large additive boost ($+20$), promoting it above semantically similar but lexically unrelated results.

This discovery layer is central to PathFinder's behavior. The agent must navigate a large search inventory where many candidate searches are close in biological meaning but differ sharply in dataset, organism, or parameter schema. The next subsection shows how this search-discovery step fits into the full strategy-construction workflow.

4.7. How PathFinder Builds a Strategy

From the user's perspective, PathFinder turns a natural-language biological request into an executable VEuPathDB strategy. Internally, this happens as a staged workflow.

PathFinder uses a single agent that has access to research, planning, and execution tools at the same time. Rather than switching between separate modes, the agent decides during each turn whether it needs to explore the catalog, reason about strategy structure, or directly create and update strategy steps. In the implementation, this workflow is orchestrated through the chat service and a unified tool registry, while validated graph-step creation is handled by a dedicated step-operations layer.

In practice, strategy construction proceeds in five stages:

1. **Interpret the request as one or more biological subgoals.** The agent first reads the prompt and breaks it into the kinds of filters the final strategy will need. For example, a request about membrane transporters in *P. vivax* may imply an organism constraint, a protein-feature constraint, and a functional annotation constraint.

2. **Discover candidate searches in the WDK catalog.** The agent then looks for different searches, datasets, and endpoints that could express each subgoal.

3. **Inspect and fill in parameters.** Once a candidate search has been chosen, the agent determines the required parameters, such as organism, dataset, threshold, or GO term. This is the stage most exposed to underspecification: the prompt may clearly imply a biological goal while still omitting the exact dataset or cutoff value needed to construct a valid step.

4. **Create and validate steps incrementally.** After selecting a search and its parameters, the agent creates the corresponding step in the strategy graph through validated tool calls. PathFinder builds or edits a strategy graph and checks parameters against

WDK search specifications before continuing. If a step fails, the error is returned in structured form so the agent can revise that step without discarding the rest of the graph.

5. **Compose steps into a full executable strategy.** Finally, the agent combines the validated steps with Boolean operators such as intersect, union, and minus, or with unary transforms such as ortholog mapping. The result is a complete strategy graph that can be executed against VEuPathDB and inspected through the interface.

4.8. Chapter Summary

This chapter described PathFinder as a tool-augmented agent for constructing executable VEuPathDB search strategies from natural-language biological questions. We introduced the strategy-construction task, outlined the system's five-layer architecture, described its tool registry, explained the semantic search mechanism that maps user intent to WDK search names, and described the step-by-step construction workflow. The following chapter turns from system construction to empirical evaluation, focusing on how prompt vagueness and strategy complexity affect construction fidelity.

CHAPTER 5 : PathFinder: Experiments on Vagueness in Strategy Construction

5.1. Gold Corpus, Vagueness Protocol, and Experimental Questions

Chapter 4 described PathFinder as a tool-augmented system for constructing VEuPathDB search strategies from natural-language inputs. This chapter turns from system design to empirical evaluation. We evaluate PathFinder on a corpus of 35 gold-standard strategies and ask how construction fidelity changes when the same biological task is expressed at three levels of prompt specificity: precise, moderate, and vague.

5.1.1. The Gold Strategy Corpus

We curate a corpus of 35 gold-standard search strategies from two categories of sources:

- **Expert-authored strategies (20):** Sourced from VEuPathDB training workshops (4 sources), research publications (7 sources), and user documentation (1 source). Workshop strategies (Athens 2-Day Workshop (VEuPathDB, 2016), Hinxton Parasite Workshop (VEuPathDB), Hinxton Fungal Workshop (VEuPathDB, 2023), and Sept 2022 GO Enrichment Workshop (VEuPathDB, 2022)) specify exact search names, parameters, Boolean operators, and expected result counts. Publication strategies are reconstructed from the VEuPathDB 2023 NAR paper (Amos et al., 2022), the 2018 FungiDB paper (Basenko et al., 2018), the 2011 WDK Strategy paper (Fischer et al., 2011), the 2010 EuPathDB portal paper (Aurrecoechea et al., 2010), the 2018 EuPathDB functional genomics paper (Aurrecoechea et al., 2017), the *T. gondii* GAMA paper (Saenz-Garcia et al., 2016), and the VectorBase updates paper (Giraldo-Calderón et al., 2015). The training module source is the VEuPathDB Strategies Training Module (VEuPathDB, 2020).
- **Public user strategies (15):** Imported from VEuPathDB's public strategy repository,

```
{
  "recordType": "transcript",
  "root": {
    "searchName": "boolean_question_GeneRecordClasses_GeneRecordClass",
    "operator": "INTERSECT",
    "primaryInput": {
      "searchName": "GenesByTransmembraneDomains",
      "parameters": {
        "organism": "[\"Plasmodium vivax P01\"]",
        "min_tm": "5",
        "max_tm": "12"
      }
    },
    "secondaryInput": {
      "searchName": "GenesByGoTerm",
      "parameters": {
        "organism": "[\"Plasmodium vivax P01\"]",
        "go_typeahead": "[\"GO:0022857\"]",
        "go_term_evidence": "[\"Curated\",\"Computed\"]",
        "go_term_slim": "No"
      }
    }
  }
}
```


Figure 5.1: Gold-standard AST for the `pvivax_tm_transporter` strategy (complexity 3). The root is an intersect node with two leaf searches: one filtering by transmembrane domain count (5–12) and one by GO term *transmembrane transporter activity* (GO:0022857). All parameter values are strings, matching the WDK REST API convention. This strategy yields 13 *P. vivax* P01 genes.

where researchers share strategies they have constructed and validated. These strategies were authored by researchers at institutions including the University of Pennsylvania, University of Notre Dame, CONICET/INGEBI, the Oswaldo Cruz Foundation, the Connecticut Agricultural Experiment Station, and others. They cover organisms and databases not well-represented in the expert-authored set, including CryptoDB, GiardiaDB, and MicrosporidiaDB. Each was verified against the WDK API before inclusion. For more details, see Appendix B.

Each gold strategy is encoded as a JSON abstract syntax tree (AST) containing the full recursive structure: search names, parameters, Boolean operators, and ortholog transforms. Strategies range from 1 to 17 AST nodes in complexity. An example is shown in Figure 5.1.

5.1.2. Complexity Distribution

Table B.2 summarizes the 35 gold strategies. We group strategies into three complexity bins based on AST node count:

- **Small (1–4 nodes):** 9 strategies. Single searches or two searches combined with one Boolean operator.
- **Medium (5–8 nodes):** 15 strategies. Multi-step workflows combining text searches, GO annotations, expression filters, protein features, and cross-species transforms.
- **Large (9+ nodes):** 11 strategies. Complex pipelines involving nested Boolean combinations, multiple expression datasets, ortholog transforms, SNP filters, phylogenetic profiles, and genomic colocation.

5.1.3. Search Types and Domain Coverage

The gold corpus exercises 31 distinct WDK base search types (44 concrete search names, since expression and phenotype searches have dataset-specific variants) across six functional categories: text and annotation, expression, protein features, genomic features, comparative genomics, and phenotype/immunology. The most frequently used searches are `GenesByText` (16 strategies), `GenesByOrthologs` (15 strategies), `GenesWithSignalPeptide` and `GenesByRNASeq` variants (12 each), and `GenesByGoTerm` and `GenesByTransmembraneDomains` (10 each). The full catalog of search types, their categories, and usage frequency is given in Table B.3.

5.1.4. Three Vagueness Tiers

For each gold strategy, we author three prompts at decreasing levels of specificity. All prompts include an instruction prohibiting clarifying questions, forcing the agent to com-

mit to a single interpretation in a one-shot setting. The agent must infer missing details from context, through literature search, or through discovery with its available tools (e.g., searching the VEuPathDB API for available datasets), rather than engaging in multi-turn parameter disambiguation with the user.

- **Precise:** Specifies the database, organism, dataset names, GO term IDs, parameter values, and logical structure, all in natural language without WDK jargon (no `INTERSECT`/`UNION` keywords or explicit search-name identifiers).
- **Moderate:** Specifies the data type, organism, and biological goal with some technical details, but omits exact dataset names and parameter values.
- **Vague:** A pure biological question with no technical details. The agent must infer the organism, database, search types, and parameters entirely from its knowledge and tool discovery.

To illustrate, the following shows the three tiers for a 3-node strategy that finds *P. vivax* membrane transporters (the underlying gold strategy used for this example was introduced in Chapter 4):

> **Precise:** "Find *P. vivax* P01 genes that have between 5 and 12 predicted transmembrane domains and are annotated with GO term transmembrane transporter activity (GO:0022857) using both curated and computed evidence."
>
> **Moderate:** "Find *P. vivax* membrane transporter genes that have multiple transmembrane domains. Look for genes with transmembrane domain predictions and GO transporter annotations."
>
> **Vague:** "Which *P. vivax* genes are likely to be membrane transporters? I want the ones that span the membrane multiple times."

This three-tier protocol lets us distinguish between moderate information loss (where the researcher knows what type of data they want but not the exact dataset) and extreme information loss (where the prompt states only a biological question and leaves the agent to infer the rest).

5.2. Experimental Setup

5.2.1. Models

We evaluate two OpenAI models: `gpt-4.1` and `gpt-4.1-mini`, both with temperature 0.

`gpt-4.1` serves as the higher-capability baseline, establishing the best performance observed in our evaluation and providing the most favorable conditions under which underspecification effects can be measured. `gpt-4.1-mini` is used to test whether the vagueness gradient generalizes across model scales and, through repeated runs, to estimate run-to-run variance via flip rates and intraclass correlation.

5.2.2. Protocol

Each of the 35 gold strategies is evaluated at all three vagueness levels. For `gpt-4.1`, we run one evaluation per strategy–vagueness pair, yielding $35 \times 3 = 105$ runs. For `gpt-4.1-mini`, we run 3 independent replicates per condition, yielding $35 \times 3 \times 3 = 315$ runs. In total, the evaluation comprises 420 runs. The three `gpt-4.1-mini` replicates are fresh independent runs with new agent sessions; the larger model is not replicated to keep the evaluation budget tractable.

Each run is a single-turn interaction: the prompt is sent, the agent builds a strategy using the 19-tool core subset (Table 4.1), and the resulting AST and gene set are compared to the gold reference.

Gene-set comparison requires executing the predicted strategy against the WDK API to retrieve gene IDs. We use a dedicated evaluation endpoint that rebuilds the predicted AST as a fresh WDK strategy and fetches the resulting gene list.

5.2.3. Metrics

We evaluate strategy quality using four primary metrics, each targeting a different aspect of the construction task, plus a supplementary structural distance measure for diagnostic analysis.

Primary Metrics

Gene-set F_1 (functional accuracy). The main test is whether the predicted and gold strategies return the same genes. We compute precision, recall, and F_1 on the gene-ID sets $\mathcal{G}_{\text{pred}}$ and $\mathcal{G}_{\text{gold}}$:

$$F_1 = \frac{2 \cdot |\mathcal{G}_{\text{pred}} \cap \mathcal{G}_{\text{gold}}|}{|\mathcal{G}_{\text{pred}}| + |\mathcal{G}_{\text{gold}}|}$$

This captures functional equivalence: two structurally different strategies that return identical genes receive $F_1 = 1.0$, while a structurally similar strategy with one wrong parameter that drastically changes the output receives $F_1 \approx 0$. We also report the perfect reconstruction rate (the fraction of runs achieving $F_1 = 1.0$).

Search accuracy. What fraction of the gold strategy's searches did the agent correctly identify? We solve the linear sum assignment problem using SciPy's `linear_sum_assignment` to find the best one-to-one pairing between predicted and gold nodes based on search-name identity. This avoids penalizing semantically irrelevant differences such as child-order swaps (which are equivalent for commutative Boolean operations). The denominator is the larger of the two non-combine node counts, $\max(|P|, |G|)$, to penalize both missing and extra searches.

Boolean combine nodes (which implement intersect, union, and minus operators) are excluded from both search accuracy and PFS computations. These nodes match trivially regardless of search-selection quality and carry no parameters, so including them would

Algorithm 1 Search accuracy via optimal node alignment

Require: Predicted AST T_p, Gold AST T_g

$P \leftarrow$ FlattenNodes(T_p) ▷ DFS collect all non-boolean nodes
$G \leftarrow$ FlattenNodes(T_g)
Initialize cost matrix $C \in \mathbb{R}^{|P| \times |G|}$ with all ones
for each $(i, j) \in |P| \times |G|$ **do**
    if `normalize`(P_i.searchName) = `normalize`(G_j.searchName) **then**
        $C[i, j] \leftarrow 0$
    end if
end for
row_ind, col_ind $\leftarrow$ `linear_sum_assignment`(C)
matched $\leftarrow |\{(r, c) : C[r, c] = 0\}|$ for $(r, c) \in$ zip(row_ind, col_ind)
return matched / max($|P|, |G|$)

inflate both metrics without reflecting genuine search-selection or parameterization skill. Only leaf searches and transform nodes are counted.

Operator accuracy. What fraction of aligned combine nodes use the same Boolean operator? This measures whether the agent selects the correct set operation at each combination step.

Parameter Fidelity Score (PFS). For non-combine node pairs that share the same search name (via optimal assignment), what fraction of their shared parameter values match? For each aligned pair, we compute:

$$\text{ParamSim}(n_p, n_g) = \frac{|\{k \in K_p \cap K_g : v_p(k) = v_g(k)\}|}{|K_p \cap K_g|}$$

where K_p, K_g are the parameter key sets and $v_p(k), v_g(k)$ are the normalized values. If there are no shared keys, ParamSim $= 0$. PFS averages this over all N_{aligned} matched pairs:

$$\text{PFS} = \frac{1}{N_{\text{aligned}}} \sum_{i=1}^{N_{\text{aligned}}} \text{ParamSim}(n_p^{(i)}, n_g^{(i)})$$

Intuitively, PFS measures what fraction of parameter values the agent gets right on steps where it selected the correct search. A score of 1.0 means all values match exactly; lower scores reflect incorrect thresholds, dataset choices, or organism selections.

Supplementary: NTED Decomposition

We compute normalized tree edit distance (NTED) via the Zhang-Shasha algorithm (Zhang and Shasha, 1989) at three label granularities: topology (all labels replaced by a constant), search (labels = search names only), and full (search names + operators + parameters, with weighted costs). The differences between layers provide a heuristic decomposition of structural error into topology, search-selection, and parameter components. Because each layer independently optimizes its own edit script, these differences are approximate attributions rather than exact causal decompositions. We retain NTED as a diagnostic tool for the failure modes analysis (Section 5.4) but do not use it as a primary hypothesis metric.

5.2.4. Statistical Analysis

Because the gold corpus is modest ($N = 35$ strategies per vagueness level), we use a non-parametric bootstrap framework:

1. Compute the metric of interest (e.g., mean F_1) on the 35 strategies.
2. Estimate its sampling distribution via non-parametric bootstrap with 10,000 resamples.
3. Report the point estimate and its 95% BCa confidence interval.

For the replicated mini model (3 runs per condition), we report the mean across replicates per strategy before bootstrapping, and additionally compute pairwise flip rates and ICC(1,1) to quantify run-to-run variance.

We also report Spearman rank correlations between search accuracy and F_1, between PFS and F_1, and between complexity and F_1.

Reproducibility. All gold strategies, prompts, evaluation scripts, and analysis code are provided as supplementary material in Appendix B.

5.3. Results

5.3.1. Overall Performance

Table 5.1 summarizes performance across the three vagueness levels on both models. The gradient is apparent: precise prompts on the larger model achieve $F_1 = 0.664$, with 51% of strategies perfectly reconstructed; moderate prompts drop to $F_1 = 0.227$ (6% perfect); vague prompts fall further to $F_1 = 0.158$ (0% perfect). The same pattern holds on the smaller model with uniformly lower absolute performance.

Vagueness	Model	Gene F_1	Search Acc.	PFS	Op. Acc.	Perfect
Precise	4.1	.664 [.524, .795]	.875 [.805, .937]	.756 [.673, .834]	.895	51%
	4.1-mini	.483 [.398, .569]	.788 [.745, .831]	.730 [.680, .778]	.728	31%
Moderate	4.1	.227 [.121, .344]	.660 [.560, .754]	.576 [.491, .659]	.808	6%
	4.1-mini	.191 [.140, .246]	.583 [.530, .639]	.537 [.484, .590]	.721	3%
Vague	4.1	.158 [.086, .241]	.443 [.355, .532]	.472 [.359, .590]	.578	0%
	4.1-mini	.098 [.062, .141]	.363 [.317, .409]	.434 [.371, .495]	.573	1%

Table 5.1: Performance across vagueness levels on both models ($N = 35$ strategies per condition; mini values are means of 3 replicates). 95% bootstrap CIs in brackets. All metrics are higher-is-better. Perfect = fraction of runs with $F_1 = 1.0$.

5.3.2. H1: Underspecification Degrades All Aspects of Strategy Construction

Functional Accuracy

Figure 5.2 shows gene-set F_1 across the three vagueness levels on the primary model. Precise prompts yield $F_1 = 0.664$ [0.524, 0.795]; moderate prompts drop to $F_1 = 0.227$ [0.121,

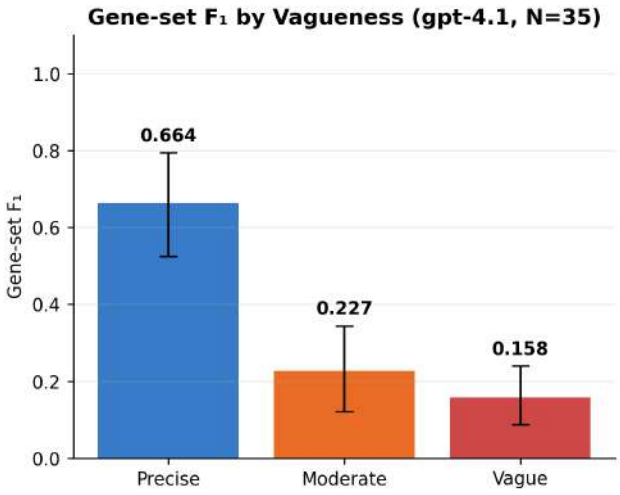


Figure 5.2: Gene-set F_1 by prompt vagueness (`gpt-4.1`, $N = 35$). Error bars show 95% bootstrap CIs. The precise–moderate gap is the dominant transition.

0.344]; vague prompts fall to $F_1 = 0.158$ [0.086, 0.241].

The pairwise analysis reveals that the degradation is concentrated at the precise-to-moderate transition. On `gpt-4.1`, the precise-to-moderate difference is $+0.437$ [0.295, 0.585] (significant), while the moderate-to-vague difference is $+0.069$ [-0.037, 0.177] (not significant at $\alpha = 0.05$). On `gpt-4.1-mini`, which has greater statistical power from 3 replicates per condition, all three pairwise differences are significant: precise-to-moderate $= +0.292$ [0.163, 0.430], moderate-to-vague $= +0.093$ [0.039, 0.154], precise–vague $= +0.385$ [0.261, 0.514].

The precise-to-moderate transition removes the identity of specific datasets (which RNA-Seq experiment, which proteomics study) while retaining the organism, data type, and biological goal. This dataset-level specification accounts for the majority of the vagueness effect: when multiple experiments of the same type exist for a single organism, the agent has no way to select the correct one without the dataset name. Threshold parameters (fold-change cutoffs, percentile ranges) are less critical, as the agent infers plausible defaults from training data (and the WDK also provides sensible defaults). The moderate-to-vague tran-

sition removes most of the remaining details, forcing the agent to infer everything from a biological question. While this adds further degradation, its marginal impact is smaller.

Component Breakdown

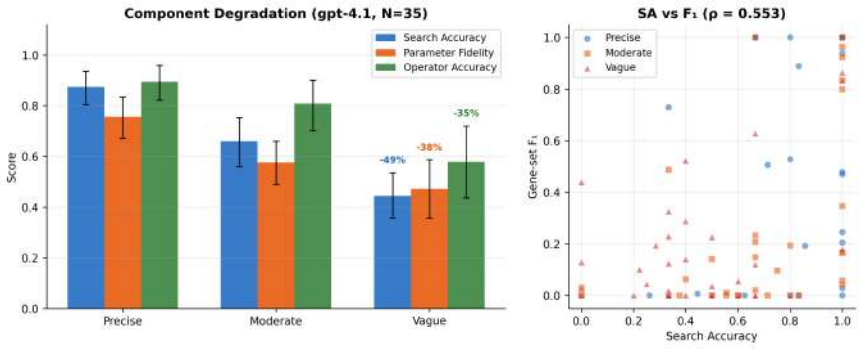


Figure 5.3: Degradation of all strategy components under increasing vagueness (`gpt-4.1`, $N = 35$). Search accuracy, parameter fidelity, and operator accuracy all decline; search accuracy is the strongest predictor of functional accuracy ($\rho = 0.574$).

Underspecification degrades every measurable aspect of strategy construction (Figure 5.3). Search accuracy drops from 87.5% to 44.3% (49% decline), parameter fidelity from 75.6% to 47.2% (38% decline), and operator accuracy from 89.5% to 57.8% (35% decline).

Search accuracy appears to be the strongest predictor of gene-set F_1: Spearman $\rho = 0.553$ ($p < 0.001$) vs. $\rho = 0.468$ ($p < 0.001$) for PFS. However, the degradation differential between SA and PFS is not statistically significant on the primary model ($+0.074$ [-0.051, 0.198]). On the replicated mini model, the differential is borderline significant ($+0.121$ [0.010, 0.227]).

Reliability

Underspecification also degrades model reliability (Figure 5.4). For each strategy at each vagueness level, we run three independent replicates and compute the gene-set F_1 between

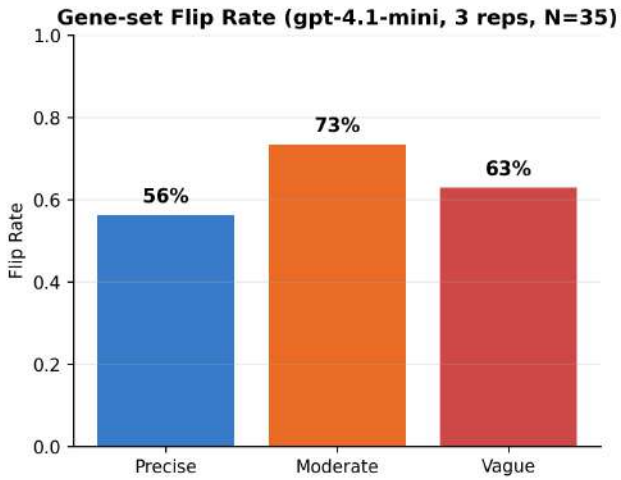


Figure 5.4: Gene-set flip rate on `gpt-4.1-mini` ($N = 35$ strategies, 3 replicates per condition). Even for precise prompts, 56% of replicate pairs produce different gene sets.

every pair of replicates' outputs. The *flip rate* measures how often these outputs diverge:

$$\text{FR}(s) = \frac{|\{(i,j) : i < j,\ F_1(\mathcal{G}_s^{(i)}, \mathcal{G}_s^{(j)}) < 1\}|}{\binom{R}{2}}$$

where R is the number of replicates and $\mathcal{G}_s^{(r)}$ is the gene set produced by replicate r for strategy s. We average FR across all strategies at each vagueness level.

Under precise prompts, 56% of replicate pairs already produce different gene sets; the rate rises to 73% under moderate and 63% under vague prompts. The lower rate for vague than moderate is a floor effect: most vague runs produce near-zero F_1, so replicates agree on returning the wrong genes rather than diverging.

To separate systematic difficulty from stochastic noise, we compute ICC(1,1), the intraclass correlation coefficient:

$$\text{ICC}(1,1) = \frac{\text{MS}_{\text{between}} - \text{MS}_{\text{within}}}{\text{MS}_{\text{between}} + (R-1) \cdot \text{MS}_{\text{within}}}$$

where $\text{MS}_{\text{between}}$ and $\text{MS}_{\text{within}}$ are the between-strategy and within-strategy mean squares

from a one-way ANOVA on F_1 scores. An ICC near 1 means replicates of the same strategy produce similar scores while different strategies produce different scores. An ICC near 0 means knowing which strategy was attempted tells you little about the outcome.

ICC drops from 0.791 (precise) to 0.747 (moderate) to 0.430 (vague). At the vague level, nearly half the observed variance has nothing to do with which strategy was attempted, so a researcher re-running the same vague prompt would get a substantially different gene set roughly half the time.

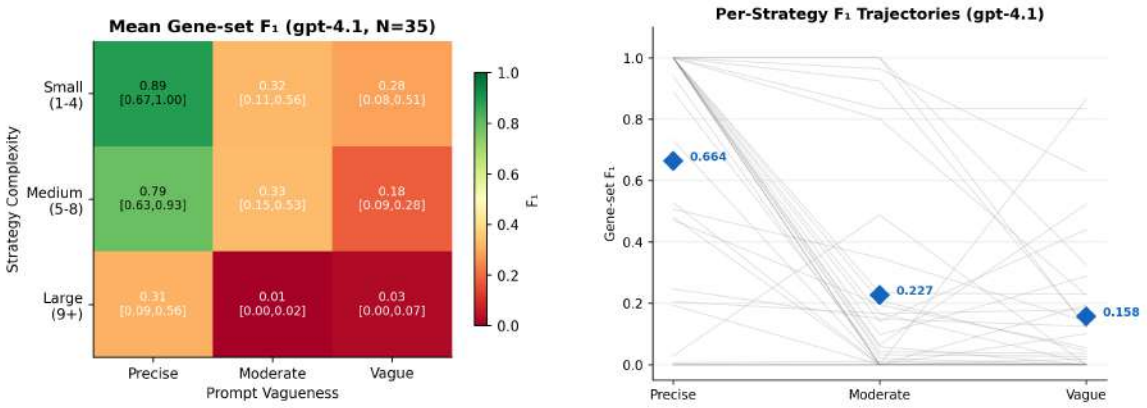
Verdict

Supported. Functional accuracy collapses from $F_1 = 0.66$ to 0.16, with the bulk of the drop at the precise-to-moderate boundary ($\Delta = 0.44$) where the prompt loses dataset names and other parameter values. Search selection, parameter fidelity, and operator choice all degrade by 35 to 49%, and run-to-run reliability drops sharply (ICC: $0.79 \rightarrow 0.43$). Search accuracy is likely the strongest predictor of gene-set F_1 ($\rho = 0.553$).

5.3.3. H2: Complexity and Vagueness Compound

Figure 5.5a shows that strategy complexity and prompt vagueness both independently reduce performance, and their combined effect is larger than either alone.

Complexity as the larger threat. Even with precise prompts, complexity dramatically reduces success. Small strategies achieve $F_1 = 0.885$ [0.663, 1.000] (8 of 9 perfectly reconstructed), medium strategies achieve 0.790 [0.631, 0.930], and large strategies decrease to 0.310 [0.088, 0.563] (only 2 of 11 perfectly reconstructed). In fact, large strategies under precise prompts ($F_1 = 0.310$) perform comparably to small strategies under vague prompts ($F_1 = 0.283$). Complexity is almost as damaging as extreme underspecification for small strategies.



(a) Mean gene-set F_1 by complexity bin $\times$ vagueness level.

(b) Per-strategy F_1 trajectories. Each gray line connects one strategy's three runs; diamonds mark group means.

Figure 5.5: H2: Complexity and vagueness compound (`gpt-4.1`, $N = 35$). (a) Both factors independently degrade performance; their combination produces near-total failure. (b) Most strategies drop steeply from precise to moderate, with several large strategies near zero across all conditions.

The compounding effect. Under vague prompts, large strategies collapse to $F_1 = 0.027$ [0.000, 0.070], while small strategies retain some accuracy ($F_1 = 0.283$ [0.082, 0.506]). This makes sense intuitively, since large strategies are already near zero under precise prompts, so additional vagueness has little further room to degrade them. Small strategies show the largest absolute drops from precise to vague (0.602 points) precisely because they start high enough to have room to fall.

The `gpt-4.1-mini` replication confirms this pattern with tighter confidence intervals: small + precise $= 0.792$ [0.663, 0.907], small + vague $= 0.148$ [0.054, 0.266], large + precise $= 0.103$ [0.030, 0.197], and large + vague $= 0.057$ [0.021, 0.106].

Verdict: Supported. Both complexity and vagueness independently degrade performance, and their combination produces near-total failure. The corpus reveals that strategy complexity is at least as damaging as prompt vagueness.

5.3.4. Model Comparison: `gpt-4.1` vs. `gpt-4.1-mini`

From Figure 5.6, we can identify three patterns:

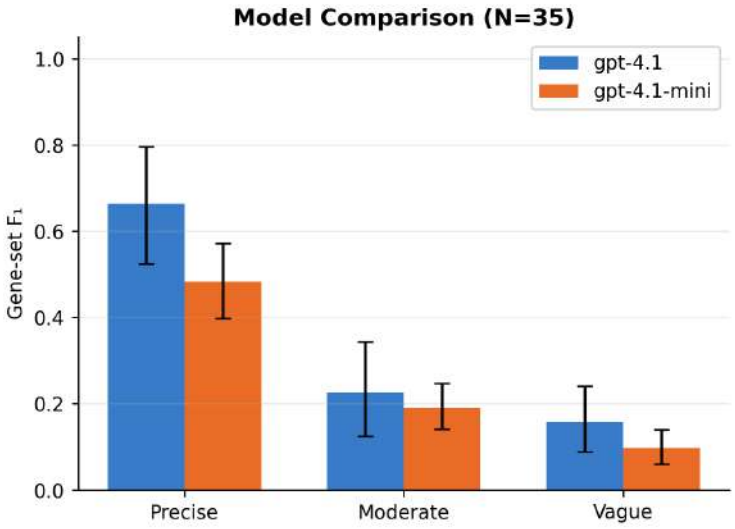


Figure 5.6: Gene-set F_1 for `gpt-4.1` ($N = 35$) vs. `gpt-4.1-mini` (mean of 3 replicates, $N = 35$) at each vagueness level. The vagueness gradient replicates on the smaller model, with the gap largest for precise prompts.

1. **The vagueness gradient replicates.** The same pattern (precise > moderate > vague) holds on both models. Both hypotheses are supported on both, with consistent effect directions and comparable effect sizes.

2. **The gap is largest for precise prompts.** The larger model achieves $F_1 = 0.664$ vs. 0.483 on precise ($\Delta = +0.181$), but the gap narrows to $+0.036$ for moderate and $+0.060$ for vague.

3. **Perfect reconstruction diverges sharply.** On precise prompts, 51% of strategies are perfectly reconstructed by the larger model vs. 31% by the smaller model. On moderate and vague prompts, both models achieve near-zero perfect reconstruction (6%/3% and 0%/1% respectively).

5.4. Failure Modes and Underspecification Effects

The central bottleneck in strategy construction is the natural-language-to-search-name mapping. VEuPathDB hosts thousands of searches with dataset-specific names like `GenesByRNASeqpfal3D7_Su_strand_specific_rnaSeq_RSRCPercentile`, and a description like "genes active during the sexual stage" could plausibly map to any of several RNA-Seq datasets for the same organism. Selecting the wrong dataset is not a minor parameter error; it is typically catastrophic, as different experiments for the same organism might share zero genes in their result sets (Figure 5.7a). Even when the agent selects the correct searches and builds the correct tree structure, a single wrong parameter value can cascade through downstream Boolean combinations and inflate the gene set by orders of magnitude (Figure 5.7b).

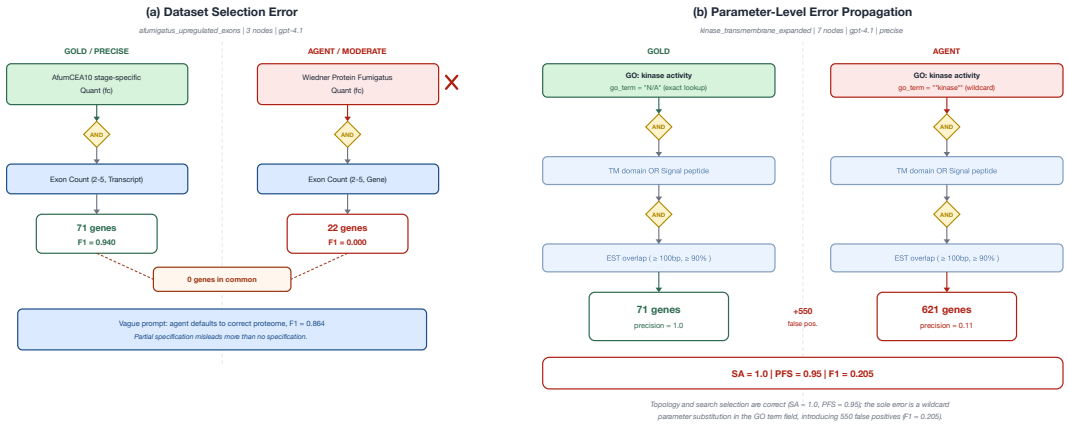


Figure 5.7: Two failure modes from real evaluation runs (`gpt-4.1`). (a) Dataset selection error: the moderate prompt causes the agent to select a different proteomics experiment (Wiedner vs. AfumCEA10) for a 3-node strategy, producing zero gene overlap despite correct tree structure. The vague prompt defaults to the correct dataset ($F_1 = 0.864$). (b) Parameter-level error propagation: the agent builds the identical 7-node tree as the gold (SA $= 1.0$, PFS $= 0.95$), but a single wildcard substitution in the GO term field (`"*kinase*"` instead of `"N/A"`) inflates the gene set from 71 to 621, producing 550 false positives ($F_1 = 0.205$).

5.4.1. Taxonomy of Failure Modes

Analysis of individual run transcripts reveals four primary failure modes:

1. **Dataset ambiguity.** When multiple searches of the same type exist for a single organism, moderate prompts that name the data type but not the exact dataset cause the agent to select the wrong experiment. This accounts for the bulk of the precise-to-moderate F_1 drop.

2. **Parameter sensitivity.** Even with perfect search selection (SA $= 1.0$) and near-perfect parameter fidelity (PFS $= 0.95$), a single wrong parameter value can cascade through downstream intersections. In the `kinase_transmembrane_expanded` example (Figure 5.7b), a wildcard `"*kinase*"` in the GO term field broadened the initial gene set, and every downstream AND operation preserved the inflation, producing 550 false positives ($F_1 = 0.205$).

3. **Operator confusion.** Under vague prompts, the agent confuses intersect, union, and minus roughly half the time (operator accuracy drops from 89.5% to 57.8% on `gpt-4.1`). The most common confusion is between intersect and union.

4. **Execution budget exhaustion.** Complex strategies (9+ nodes) with hard-to-parameterize searches consume the single-turn execution budget on validation-error retries, leaving later steps unbuilt. `possible_vaccine_candidate` (17 nodes) builds 9 of 17 steps before exhausting its budget, retaining 324 genes instead of filtering to 7 ($F_1 = 0.006$).

5.4.2. Structural vs. Functional Accuracy

A notable asymmetry exists between structural and functional accuracy. Several runs produce structurally different trees that nonetheless return the correct gene set. For example,

`agambiae_hemocyte_secreted` under moderate prompting on `gpt-4.1` uses a different combination of searches from the gold standard but achieves $F_1 = 1.0$: the agent found an alternative path through the WDK search space that converges on the same gene set.

More common is the reverse: 31 runs (across both models) have structurally similar strategies (NTED < 0.3) but low F_1 (< 0.3), indicating that small parameter differences can produce dramatically different gene sets.

5.5. Discussion

5.5.1. Summary of Findings

The evaluation yields four principal findings.

First, the tool-augmented agent achieves high fidelity on simple and medium-complexity strategies when provided with precise prompts: 18 of 35 strategies (51%) are perfectly reconstructed, rising to 67% for the 24 strategies with fewer than 9 AST nodes. This indicates that the tool-calling infrastructure is sufficient for the task; the primary limiting factor is the content of the prompt.

Second, the degradation induced by underspecification is concentrated at the precise-to-moderate boundary ($\Delta F_1 = 0.437$ [0.295, 0.585]), where the prompt loses dataset-level identifiers and parameter values while retaining the organism, data type, and biological goal. This suggests that dataset-level specification contributes more to strategy correctness than concept-level specification, especially for `RNA-Seq` and `Microarray` searches.

Third, underspecification degrades all measurable components of strategy construction. Search accuracy, parameter fidelity, and operator accuracy decline by 49%, 38%, and 35% respectively from precise to vague prompts. Search accuracy is the strongest predictor of functional accuracy (Spearman $\rho = 0.553$ vs. 0.468 for PFS), though the degradation rates across components are comparable rather than concentrated in a single metric.

Fourth, strategy complexity is a major independent factor. Large strategies (9+ nodes) achieve $F_1 = 0.310$ under precise prompts, comparable to small strategies under vague prompts ($F_1 = 0.283$). Six of 11 large strategies achieve $F_1 < 0.2$ even with precise prompts, which shows both LLM capability limits (difficulty selecting rare searches, error compounding across sequential decisions) and system design constraints (having a fixed execution budget, for example) that disproportionately affect complex strategies.

5.5.2. Implications

The degradation at the precise-to-moderate boundary has a practical consequence: for researchers using tool-augmented agents for genomics queries, specifying the exact dataset name and parameter thresholds yields substantially greater fidelity than providing only the organism and data type. Conversely, the relatively small marginal effect of the moderate-to-vague transition suggests that broad biological framing contributes less to strategy quality than specific methodological details.

The model comparison reveals that scaling from `gpt-4.1-mini` to `gpt-4.1` yields the largest benefit on precise prompts ($\Delta F_1 = +0.181$) and a negligible benefit on vague prompts ($\Delta F_1 = +0.060$). Within the models evaluated here, scaling from `gpt-4.1-mini` to `gpt-4.1` does not eliminate the degradation caused by underspecification.

The high run-to-run variance observed in the replicated evaluation (flip rates of 56–73%, ICC as low as 0.43 under vague prompts) has methodological implications for future work, where replicated designs with variance reporting should be considered standard practice.

5.5.3. PathFinder-Specific Limitations

- **Replicate asymmetry:** The primary model (`gpt-4.1`) has one run per condition; variance estimates come from 3 replicates on the mini model. We treat the primary model as the capability ceiling and rely on the replicated mini evaluation for statistical

inference about variance and stability. The high flip rates (56–73%) suggest that single-run estimates should be interpreted with caution.

- **Live database evaluation:** Evaluating against the live VEuPathDB API introduces a time dependence: gene counts and search availability can shift as databases are updated. We mitigate this by re-seeding all gold strategies against the current database immediately before evaluation, amending parameters where the original values no longer produce results. This ensures that gold and predicted strategies are evaluated against the same database snapshot, but means our gold strategies are not exact reproductions of the original publications.

- **No interactive disambiguation:** All evaluations force single-turn interaction. In practice, PathFinder supports multi-turn conversations where the researcher can refine the strategy. Our results represent a lower bound on achievable accuracy.

Broader limitations shared across the thesis, including the reliance on a single LLM provider, the lack of human-expert baselines, and the challenges of benchmark construction for emerging tool-calling systems, are discussed in Chapter 6.

5.6. Chapter Summary

This chapter evaluated PathFinder on a 35-strategy corpus under three levels of prompt vagueness. The results show a strong and replicated vagueness gradient, with the largest drop occurring when prompts lose dataset-level specificity. They also show that strategy complexity is a major independent source of failure, and that underspecification degrades not only functional gene-set accuracy but also search selection, parameter fidelity, operator choice, and run-to-run reliability.

CHAPTER 6 : Discussion and Conclusion

The preceding case-study chapters present two case studies of tool-augmented LLM agents in materials science (LAMINA, Chapters 2 and 3) and pathogen genomics (PathFinder, Chapters 4 and 5). Both studies vary prompt specificity under controlled conditions and measure the resulting degradation in output quality and reliability. This chapter synthesizes the findings, identifies failure patterns that generalize across domains, derives design recommendations for scientific tool-calling agents, and discusses shared limitations and directions for future work.

6.1. Cross-Case Synthesis

Table 6.1 summarizes the two case studies along key dimensions.

Dimension	**LAMINA**	**PathFinder**
Domain	Materials science	Pathogen genomics
Task type	Single-turn feasibility judgment	Multi-step strategy construction
Output	Ordinal label $\{-2, \ldots, +2\}$	Recursive search tree + gene set
Tool count	50	44 (19 in eval)
Benchmark size	37 claims	35 gold strategies
Vagueness levels	2 (precise, vague)	3 (precise, moderate, vague)
Models evaluated	gpt-4o-mini, gpt-4o, gpt-5.1	gpt-4.1, gpt-4.1-mini
Total runs	37 $\times$ configs	420 (105 + 315 replicated)
Primary metric	QWK	Gene-set F_1
Reliability metric	FR / QWFR	Flip rate / ICC(1,1)

Table 6.1: Side-by-side comparison of the two case studies.

Despite the differences in domain, task structure, output type, and evaluation metric, the two studies converge on a common finding: **among the factors investigated in this thesis (prompt underspecification, model size, routing policy, and task complexity), underspecification is the dominant source of output instability.**

Same root cause. In both systems, the instability localizes to the natural-language-to-tool-invocation mapping. In LAMINA, vague quantifiers and underspecified comparators map to different numeric tool inputs, producing label flips (FR $= 0.43$, QWFR $= 0.14$, both with 95% CIs entirely above zero). In PathFinder, missing dataset names and parameter values cause the agent to select wrong searches and parameterize them incorrectly, collapsing gene-set F_1 from 0.664 to 0.158.

Different manifestations. The underspecification effect surfaces differently in each task type. LAMINA's single-turn judgment task produces *label flips*: the same claim yields different ordinal verdicts depending on how vague phrases are resolved. PathFinder's multi-step construction task produces *strategy collapse*: the agent builds structurally and functionally incorrect search trees, with all measurable components (search accuracy, parameter fidelity, operator accuracy) degrading by 35–49%. The distinction reflects the difference between a classification task (where each claim is an independent prediction) and a construction task (where errors in early steps propagate through downstream Boolean combinations).

Error compounding in multi-step tasks. PathFinder's multi-step structure introduces a compounding mechanism absent from LAMINA. Each step in a strategy tree is an independent opportunity for the agent to select the wrong search or set wrong parameters, and errors in early steps propagate through downstream intersections and unions. Large strategies (9+ nodes) under precise prompts achieve $F_1 = 0.310$, comparable to small strategies under vague prompts ($F_1 = 0.283$). Under vague prompts, large strategies collapse to $F_1 = 0.027$ (basically 0). LAMINA's single-turn task has no such compounding: each claim is evaluated independently, so errors do not cascade.

Model scale does not currently compensate. Within the range of models evaluated here, increasing model size does not eliminate the underspecification effect. In LAMINA, the QWK gap between `gpt-5.1` and `gpt-4o-mini` shrinks from 0.235 to 0.081 with tools, but neither model overcomes the ambiguity-driven flip rate. In PathFinder, the F_1 gap between `gpt-4.1` and `gpt-4.1-mini` is largest for precise prompts ($\Delta = +0.181$) and negligible for vague prompts ($\Delta = +0.060$). Scaling helps when the prompt provides enough information for the larger model to exploit; it does not help when the prompt leaves the correct tool invocation underdetermined.

6.2. Common Failure Patterns

Three failure patterns appear in both systems, mapped to the taxonomy of underspecification defined in Chapter 1.

Selection errors. The agent invokes an inappropriate tool or neglects a necessary one. In LAMINA, the No-Policy router exhibits a miss-rate of 8.1%, where required tool families are never invoked for a given claim. In PathFinder, search accuracy drops from 87.5% to 44.3% as prompts become more vague, with the dominant failure being dataset ambiguity: when multiple RNA-Seq or proteomics experiments exist for the same organism, the agent selects the wrong one without the dataset name. Selection errors correspond to *ambiguous tool choice and data sources* (category 5) in the taxonomy.

Parameterization instability. The agent selects the correct tool but provides wrong or inconsistent arguments. In LAMINA, "a few percent" Zn maps to 2%, 3%, or 5% across runs, straddling a phase-formation boundary and flipping the verdict by four ordinal levels. In PathFinder, a single wildcard substitution in a GO term field (`"*kinase*"` instead of `"N/A"`) inflates a gene set from 71 to 621, producing 550 false positives ($F_1 = 0.205$)

despite otherwise perfect structural alignment (SA $= 1.0$, PFS $= 0.95$). Parameterization errors correspond to *ambiguous quantities and thresholds* (category 4) and *missing constraints and operating assumptions* (category 2).

Verbalization drift and operator confusion. In LAMINA, the model's final explanation sometimes deviates from the tool outputs, misreading phase fractions or understating dominant phases, effectively changing the predicted label without changing any tool call. In PathFinder, the analogous failure is operator confusion: under vague prompts, the agent confuses INTERSECT, UNION, and MINUS roughly half the time (operator accuracy drops from 89.5% to 57.8%). Both are instances of the model's natural-language reasoning diverging from the structured tool semantics. These correspond to *missing objectives and success criteria* (category 1), where the prompt does not constrain what a satisfactory answer looks like.

6.3. Design Recommendations

The cross-case analysis motivates four design principles for scientific tool-calling agents. Each is grounded in specific failure modes observed across both systems.

6.3.1. Disambiguation Protocols

When a prompt is underspecified, the agent should detect the ambiguity and request clarification rather than silently choosing a default. Both studies force single-turn interaction as a methodological constraint, producing results that represent a lower bound on achievable accuracy. In practice, a single clarifying question ("Which RNA-Seq experiment: Su et al. or Bartfai et al.?") could resolve the dataset ambiguity that accounts for the bulk of PathFinder's precise-to-moderate F_1 drop ($\Delta = 0.437$). Prior work supports this: Aliannejadi et al. (2019) show that a single clarifying question improves retrieval precision by

170% in conversational search.

Implementation requires two capabilities: (1) a classifier, a model, a heuristic, or another framework that detects when a prompt maps to multiple plausible tool invocations, and (2) a structured follow-up mechanism that resolves the missing fields. This second step need not rely only on presenting ranked alternatives to the user. The model can also generate the clarifying question itself; for example, STaR-GATE (Andukuri et al., 2024) explores teaching language models to ask useful clarifying questions. When the expected fields are already defined in the tool schema, this follow-up process begins to resemble a slot-filling problem: the system can identify which required arguments are underspecified and ask specifically for those. LAMINA's Policy router partially addresses (1) by routing claims through decision rules, achieving a 0% miss-rate (H2b), though it risks overfitting; PathFinder's semantic search catalog (`search_for_searches`) provides part of the foundation for (2) by surfacing ranked candidates that could either be presented as options or used to support a model-generated clarification step.

6.3.2. Catalog Grounding

Tool selection should be grounded in a structured catalog rather than relying on the model's parametric knowledge of tool names and capabilities. PathFinder's search discovery mechanism uses semantic retrieval (nomic-embed-text-v1.5 embeddings with keyword boosting) over the WDK search catalog, and achieves 87.5% search accuracy under precise prompts. The remaining errors concentrate in cases where the catalog contains multiple semantically similar entries (e.g., several RNA-Seq datasets for the same organism) that differ only in the specific experiment.

6.3.3. Incremental Validation

Strategy construction should include intermediate validation steps that catch errors before they propagate. PathFinder implements an auto-build hook, which materializes every graph mutation on VEuPathDB immediately, providing the agent with result counts after each step. When a step returns zero results, the agent receives immediate feedback and can revise its parameters.

The value of incremental validation is visible in the structural-vs-functional accuracy asymmetry: 31 runs across both PathFinder models produce strategies with low structural distance (NTED < 0.3) but low functional accuracy ($F_1 < 0.3$), indicating that small parameter errors cascade through downstream operations. Intermediate validation at each step could catch these cascading errors early.

LAMINA's evaluation pipeline does not include intermediate validation (claims are assessed in a single pass), which limits the system's ability to self-correct.

6.3.4. Replicated Evaluation Designs

Single-run evaluations of tool-calling agents are insufficient for drawing reliable conclusions. PathFinder's replicated evaluation reveals flip rates of 56–73% and ICC values as low as 0.43 under vague prompts, meaning that a researcher re-running the same prompt would get a substantially different gene set roughly half the time. LAMINA's flip rate analysis shows comparable instability: FR $= 0.43$ across alternate interpretations of vague claims.

Replicated designs with variance reporting should be standard practice for evaluating scientific agents. At minimum, evaluations should report bootstrap confidence intervals on primary metrics and measure run-to-run consistency via ICC or an equivalent reliability statistic.

6.4. Shared Limitations

Several limitations apply to both case studies beyond those discussed in their respective chapters.

Single LLM provider. Both systems evaluate only OpenAI models: `gpt-4o-mini`, `gpt-4o`, and `gpt-5.1` for LAMINA; `gpt-4.1` and `gpt-4.1-mini` for PathFinder. While the vagueness gradient replicates across model sizes within each study, cross-provider generalization remains untested. Differences in tool-calling conventions, function-calling formats, and training data could alter the magnitude of the underspecification effect.

No human-expert baselines. Neither study compares agent performance against timed human expert sessions. LAMINA's gold labels come from expert consensus within the DARPA SciFy program; PathFinder's gold strategies come from published workshop exercises and research papers. A human-expert baseline would establish the ceiling for each task and contextualize the agent's performance relative to domain specialists. Collecting such baselines requires substantial domain expertise and is left to future work.

Small benchmarks. LAMINA evaluates on 37 claims and PathFinder on 35 gold strategies. Both benchmarks are large enough to detect the dominant vagueness effect (LAMINA: H3a supported; PathFinder: H1 and H2 supported with significant CIs), but several secondary hypotheses in LAMINA (H1, H2a, H4, H5) are marked "not supported" primarily due to insufficient statistical power rather than absent effects. Expanding both benchmarks is constrained by the availability of published, reproducible tasks with known correct outputs.

Single prompt author. All alternate prompts in LAMINA (37 precise/vague claim expansions for LAMINA) and the 105 prompts for PathFinder (35 $\times$ 3 levels) were authored by a single researcher. The vagueness tiers are intuitively grounded in the taxonomy from Chapter 1 but not formally operationalized with inter-rater agreement. Different prompt authors might produce systematically different vagueness distributions, and the results may partially reflect the author's phrasing conventions.

6.5. Future Work

6.5.1. Cross-Provider Replication

The most direct extension is replicating both studies on models from other providers. PathFinder's architecture already supports Anthropic (Claude), Google (Gemini), and local models via Ollama, and LAMINA's Kani-based design permits the same provider swaps. Cross-provider replication would test whether the underspecification effect is a property of the task or an artifact of OpenAI's tool-calling implementation.

6.5.2. Interactive Disambiguation

Both studies enforce single-turn interaction as a methodological control. A natural follow-up is to measure the effect of allowing the agent to ask clarifying questions. PathFinder already supports multi-turn conversations; LAMINA could be extended to insert a disambiguation step before tool invocation. The key measurement would be the reduction in flip rate and the increase in F_1 or QWK as a function of the number of clarifying turns.

6.5.3. Larger, Community-Contributed Benchmarks

Expanding the evaluation corpora would tighten confidence intervals on secondary hypotheses and enable finer-grained analysis (e.g., per-search-type accuracy, per-domain per-

formance). For PathFinder, the VEuPathDB community maintains hundreds of public strategies that could be validated and added to the corpus. For LAMINA, the SciFy program continues to produce new claims across materials science subdomains. Community contribution would also address the single-prompt-author limitation by introducing diverse phrasing conventions.

6.5.4. Fine-Tuning on Tool Traces

Both systems generate detailed execution traces (tool calls, arguments, return values) that could serve as supervised training data. Fine-tuning a smaller model on traces from successful runs could improve tool selection and parameterization without requiring a frontier-class base model. PathFinder's 420 logged runs and LAMINA's complete execution histories provide a starting dataset.

6.5.5. Human Studies

Controlled user studies with domain experts (parasitologists for PathFinder, materials scientists for LAMINA) would establish human-expert baselines and measure how interactive disambiguation affects both the quality of the output and the researcher's trust in the system. Such studies would also reveal whether domain experts naturally provide precise or vague prompts, informing the practical relevance of each vagueness tier.

6.5.6. System Improvements and Integration

PathFinder is under active development toward integration with VEuPathDB's production infrastructure. Ongoing work includes improvements to search discovery (better disambiguation of semantically similar datasets), context summarization for long conversations, known bug fixes in parameter handling and gene ID resolution, and architectural enhancements informed by the failure modes identified in this evaluation. These improvements

target the specific bottlenecks revealed by the evaluation: dataset ambiguity at the precise-to-moderate boundary, parameter sensitivity in multi-step strategies, and execution budget exhaustion for complex strategies. Measuring the impact of these system-level changes on the same benchmark would provide a controlled before-and-after comparison and quantify the value of engineering effort relative to model scaling.

6.6. Conclusion

This thesis asked how prompt underspecification shapes the behavior of tool-calling LLM agents in scientific workflows. Across both case studies, and among the factors investigated in this thesis (prompt underspecification, model size, routing policy, and task complexity), underspecification is the dominant source of output instability, localizing to the natural-language-to-tool-invocation mapping rather than to tool execution and compounding in multi-step tasks where each step introduces an independent opportunity for error.

Four research questions motivated the investigation:

RQ1 (quality degradation): Prompt underspecification degrades output quality substantially. The effect is larger than model scale (PathFinder: $\Delta F_1 = 0.437$ for precise-to-moderate vs. $\Delta F_1 = 0.181$ for mini-to-full at precise) and larger than task complexity for small-to-medium strategies. Only for large strategies does complexity rival vagueness in magnitude.

RQ2 (component sensitivity): All components of tool use degrade under underspecification. In PathFinder, search accuracy, parameter fidelity, and operator accuracy decline by 49%, 38%, and 35% respectively. Search accuracy is the strongest single predictor of functional accuracy ($\rho = 0.553$), but no single component is the exclusive bottleneck.

RQ3 (reliability): Underspecification degrades reliability and accuracy. PathFinder's flip rates range from 56% to 73%, and ICC drops from 0.791 (precise) to 0.430 (vague).

LAMINA's flip rate of 0.43 on vague claims confirms the same pattern in a different domain. A researcher re-running the same vague prompt should expect a substantially different output.

RQ4 (generalizability): The effects replicate across two scientific domains (materials science, pathogen genomics), two task types (single-turn judgment, multi-step construction), five models (`gpt-4o-mini`, `gpt-4o`, `gpt-5.1`, `gpt-4.1`, and `gpt-4.1-mini`), and two evaluation paradigms (ordinal agreement, gene-set overlap). The root cause (underspecified natural-language-to-tool mapping) and the dominant failure patterns (selection errors, parameterization instability, and divergence between natural language reasoning and structured tool semantics) are shared. Cross-provider replication remains an open question.

The practical implication is clear: tool-calling agents do not eliminate the translation burden between scientific intent and computational execution. They shift more of that burden into the natural-language interface. In PathFinder, omitting the dataset name can collapse gene-set F_1 from 0.664 to 0.227, and in LAMINA, resolving a vague phrase like "a few percent" differently can flip the final verdict across multiple ordinal levels.

For researchers, specifying the exact dataset and parameter thresholds yields substantially greater fidelity than describing the scientific question in general terms. The concentration of degradation at the precise-to-moderate boundary in PathFinder suggests that eliciting even one additional detail from the user could recover a substantial portion of the F_1 gap. For system builders, the findings motivate disambiguation protocols, incremental validation, and replicated evaluation as architectural priorities.

Within the systems and models evaluated here, the translation problem that motivates this thesis is not solved by scaling models or expanding tool registries alone. It requires systems that detect when a prompt is underspecified and engage the researcher in structured clarification before committing to a tool-invocation path. Until then, the gap between what a researcher means and what an agent executes will remain a central bottleneck in AI-assisted

scientific workflows.

APPENDIX A : LAMINA Supplementary Materials

A.1. System Execution Examples

The following examples illustrate LAMINA's tool-augmented query processing across different materials science domains. Each example shows the user query, tool invocations with parameters, tool outputs, and the final model response.

A.1.1. Example 1: Phase Equilibrium Query (CALPHAD)

User Query

```
What phases are stable in Al20Zn80 at 870K?
```

Tool Call: calculate_equilibrium_at_point


```
{
  "composition": "Al20Zn80",
  "temperature": 870
}
```


Tool Response


```
{
  "success": true,
  "error": null,
  "error_type": null,
  "metadata": {
    "handler": "calphad",
    "function": "calculate_equilibrium_at_point",
    "timestamp": "2025-12-02T21:14:00.218892Z",
    "version": "0.0.1",
    "duration_ms": 1633.206844329834
  },
  "citations": [
    "pycalphad"
  ],
  "data": {
    "temperature": {
```

```
      "temperature_K": 870,
      "temperature_C": 596.85
    },
    "composition": {
      "input_string": "Al20Zn80",
      "composition_type": "atomic",
      "atomic_fractions": {
        "AL": 0.2,
        "ZN": 0.8
      },
      "atomic_percent": {
        "AL": 20,
        "ZN": 80
      },
      "elements": [
        "AL",
        "ZN"
      ],
      "system": "AL-ZN"
    },
    "phases": [
      {
        "phase_name": "LIQUID",
        "phase_name_raw": "LIQUID",
        "fraction": 1,
        "fraction_percent": 100,
        "composition_atomic_fraction": {
          "AL": 0.20000000000000243,
          "ZN": 0.7999999999999976
        },
        "composition_atomic_percent": {
          "AL": 20,
          "ZN": 80
        }
      }
    ],
    "total_fraction": 1,
    "n_phases": 1
  },
  "has_image": false,
  "has_html": false,
  "notes": [
    "Equilibrium calculated using robust compute_equilibrium utility",
    "Found 1 stable phase(s)",
    "Total phase fraction: 100.00%"
  ],
  "confidence": "high"
}
```

Model Response

At a temperature of 870 K, the stable phase in the composition of Al20Zn80 is the LIQUID phase, which constitutes 100% of the system at this condition.

This result indicates that at this temperature and composition, the aluminum-zinc system is entirely in the liquid state.

Citations: Results based on CALPHAD calculations (pycalphad tool).

A.1.2. Example 2: Battery Electrode Analysis (Electrochemistry)

User Query

```
What is the average voltage of an AlMg anode in a lithium bat-
tery?
```

Tool Call: search_battery_electrodes


```
{
  "formula": "AlMg",
  "max_entries": 1
}
```


Tool Response


```
{
  "success": true,
  "error": null,
  "error_type": null,
  "metadata": {
    "handler": "electrochemistry",
    "function": "search_battery_electrodes",
    "timestamp": "2025-12-02T21:20:09.181385Z",
    "version": "0.0.1",
    "duration_ms": 442.02589988708496
  },
  "citations": [
    "Materials Project",
```

```
    "pymatgen"
  ],
  "data": {
    "count": 1,
    "electrodes": [
      {
        "battery_id": null,
        "material_id": null,
        "input_formula": "AlMg",
        "host_formula": "MgAl",
        "formula": "MgAl",
        "formula_discharge": null,
        "formula_charge": null,
        "working_ion": "Li",
        "average_voltage": 0.18604090416666383,
        "max_voltage_step": 0.34988905009800453,
        "min_voltage": 0.051608643333275146,
        "max_voltage": 0.34988905009800453,
        "capacity_grav": 1567.7213201516392,
        "capacity_basis": "per gram of host (MgAl)",
        "capacity_vol": null,
        "energy_grav": 291.66029188236683,
        "energy_vol": null,
        "fracA_charge": null,
        "fracA_discharge": null,
        "framework": "Al-Mg",
        "source": "computed_from_phase_diagram",
        "diagnostics": {
          "chemsys": "Al-Li-Mg",
          "x_max_scanned": 3,
          "x_host_max": 1.5,
          "x_mix_at_x_host_max": 0.75,
          "dx": 0.05,
          "min_voltage_cutoff": 0.02,
          "n_steps": 30,
          "n_host_atoms_per_fu": 2,
          "host_molar_mass_g_per_mol": 51.2865386,
          "plating_starts_at_x_host": null,
          "note": "x_host = Li per host atom; x_mix = mole fraction of Li in mixture"
        },
        "voltage_profile_steps": [
          {
            "x_host_1": 0,
            "x_host_2": 0.05,
            "x_mix_1": 0,
            "x_mix_2": 0.09090909090909091,
            "V": 0.34988905009800453
          },
          {
            "x_host_1": 0.05,
            "x_host_2": 0.1,
            "x_mix_1": 0.09090909090909091,
```

```
      "x_mix_2": 0.16666666666666669,
      "V": 0.3349205133333206
    },
    {
      "x_host_1": 0.1,
      "x_host_2": 0.15000000000000002,
      "x_mix_1": 0.16666666666666669,
      "x_mix_2": 0.23076923076923078,
      "V": 0.33454540339245553
    },
    {
      "x_host_1": 0.15000000000000002,
      "x_host_2": 0.2,
      "x_mix_1": 0.23076923076923078,
      "x_mix_2": 0.28571428571428575,
      "V": 0.32854364433840216
    },
    {
      "x_host_1": 0.2,
      "x_host_2": 0.25,
      "x_mix_1": 0.28571428571428575,
      "x_mix_2": 0.3333333333333333,
      "V": 0.3285436443384424
    },
    {
      "x_host_1": 0.25,
      "x_host_2": 0.30000000000000004,
      "x_mix_1": 0.3333333333333333,
      "x_mix_2": 0.37500000000000006,
      "V": 0.3285436443383852
    },
    {
      "x_host_1": 0.30000000000000004,
      "x_host_2": 0.35000000000000003,
      "x_mix_1": 0.37500000000000006,
      "x_mix_2": 0.4117647058823529,
      "V": 0.32854364433844213
    },
    {
      "x_host_1": 0.35000000000000003,
      "x_host_2": 0.4,
      "x_mix_1": 0.4117647058823529,
      "x_mix_2": 0.4444444444444445,
      "V": 0.3285436443383894
    },
    {
      "x_host_1": 0.4,
      "x_host_2": 0.45,
      "x_mix_1": 0.4444444444444445,
      "x_mix_2": 0.4736842105263158,
      "V": 0.32854364433842326
    },
```

```
        {
          "x_host_1": 0.45,
          "x_host_2": 0.5,
          "x_mix_1": 0.4736842105263158,
          "x_mix_2": 0.5,
          "V": 0.32094591675421785
        },
        {
          "x_host_1": 0.5,
          "x_host_2": 0.55,
          "x_mix_1": 0.5,
          "x_mix_2": 0.5238095238095238,
          "V": 0.2153375033333175
        },
        {
          "x_host_1": 0.55,
          "x_host_2": 0.6000000000000001,
          "x_mix_1": 0.5238095238095238,
          "x_mix_2": 0.5454545454545455,
          "V": 0.21533750333331195
        },
        {
          "x_host_1": 0.6000000000000001,
          "x_host_2": 0.65,
          "x_mix_1": 0.5454545454545455,
          "x_mix_2": 0.5652173913043479,
          "V": 0.21533750333334573
        },
        {
          "x_host_1": 0.65,
          "x_host_2": 0.7000000000000001,
          "x_mix_1": 0.5652173913043479,
          "x_mix_2": 0.5833333333333333,
          "V": 0.21533750333330695
        },
        {
          "x_host_1": 0.7000000000000001,
          "x_host_2": 0.75,
          "x_mix_1": 0.5833333333333333,
          "x_mix_2": 0.6,
          "V": 0.21307804910521505
        },
        {
          "x_host_1": 0.75,
          "x_host_2": 0.8,
          "x_mix_1": 0.6,
          "x_mix_2": 0.6153846153846154,
          "V": 0.12357503795296754
        },
        {
          "x_host_1": 0.8,
          "x_host_2": 0.8500000000000001,
```

```
      "x_mix_1": 0.6153846153846154,
      "x_mix_2": 0.6296296296296297,
      "V": 0.11858809166665696
    },
    {
      "x_host_1": 0.8500000000000001,
      "x_host_2": 0.9,
      "x_mix_1": 0.6296296296296297,
      "x_mix_2": 0.6428571428571429,
      "V": 0.11858809166664945
    },
    {
      "x_host_1": 0.9,
      "x_host_2": 0.9500000000000001,
      "x_mix_1": 0.6428571428571429,
      "x_mix_2": 0.6551724137931034,
      "V": 0.1185880916667047
    },
    {
      "x_host_1": 0.9500000000000001,
      "x_host_2": 1,
      "x_mix_1": 0.6551724137931034,
      "x_mix_2": 0.6666666666666666,
      "V": 0.11858809166663946
    },
    {
      "x_host_1": 1,
      "x_host_2": 1.05,
      "x_mix_1": 0.6666666666666666,
      "x_mix_2": 0.6774193548387097,
      "V": 0.06785313833330026
    },
    {
      "x_host_1": 1.05,
      "x_host_2": 1.1,
      "x_mix_1": 0.6774193548387097,
      "x_mix_2": 0.6875,
      "V": 0.06785313833336576
    },
    {
      "x_host_1": 1.1,
      "x_host_2": 1.1500000000000001,
      "x_mix_1": 0.6875,
      "x_mix_2": 0.696969696969697,
      "V": 0.067853138333348
    },
    {
      "x_host_1": 1.1500000000000001,
      "x_host_2": 1.2000000000000002,
      "x_mix_1": 0.696969696969697,
      "x_mix_2": 0.7058823529411765,
      "V": 0.06785313833335799
```

```
        },
        {
          "x_host_1": 1.2000000000000002,
          "x_host_2": 1.25,
          "x_mix_1": 0.7058823529411765,
          "x_mix_2": 0.7142857142857143,
          "V": 0.06785313833328056
        },
        {
          "x_host_1": 1.25,
          "x_host_2": 1.3,
          "x_mix_1": 0.7142857142857143,
          "x_mix_2": 0.7222222222222222,
          "V": 0.05160864333335397
        },
        {
          "x_host_1": 1.3,
          "x_host_2": 1.35,
          "x_mix_1": 0.7222222222222222,
          "x_mix_2": 0.7297297297297297,
          "V": 0.05160864333338062
        },
        {
          "x_host_1": 1.35,
          "x_host_2": 1.4000000000000001,
          "x_mix_1": 0.7297297297297297,
          "x_mix_2": 0.736842105263158,
          "V": 0.051608643333325106
        },
        {
          "x_host_1": 1.4000000000000001,
          "x_host_2": 1.4500000000000002,
          "x_mix_1": 0.736842105263158,
          "x_mix_2": 0.7435897435897436,
          "V": 0.051608643333275146
        },
        {
          "x_host_1": 1.4500000000000002,
          "x_host_2": 1.5,
          "x_mix_1": 0.7435897435897436,
          "x_mix_2": 0.75,
          "V": 0.051608643333328666
        }
      ]
    }
  ],
  "query": {
    "formula": "AlMg",
    "working_ion": "Li"
  }
},
"has_image": false,
```

```
  "has_html": false,
  "notes": [
    "Found 1 electrode materials (computed from convex hull)",
    "Voltages are reported vs. the working ion (e.g., Li/Li+)",
    "capacity_grav is in mAh/g, energy_grav is in Wh/kg",
    ""
  ],
  "confidence": "high"
}
```


Model Response

> The average voltage of an AlMg anode in a lithium battery is approximately 0.186 V.
>
> This value is determined in relation to the working ion (Li), and it reflects the electrochemical performance of the AlMg electrode under the specified conditions.
>
> Citations: Results from the Materials Project database.

A.1.3. Example 3: Bulk Modulus Query (Materials Project)

User Query

```
What is the bulk modulus of silver?
```

Tool Call: get_elastic_properties


```
{
  "chemsys": "Ag"
}
```


Tool Output


```
{
  "tool_name": "get_elastic_properties",
  "result": {
```

```
  "success": true,
  "error": null,
  "error_type": null,
  "metadata": {
    "handler": "materials",
    "function": "get_elastic_properties",
    "timestamp": "2025-12-02T21:34:48.258866Z",
    "version": "0.0.1",
    "duration_ms": 1082.4499130249023
  },
  "citations": [
    "Materials Project"
  ],
  "data": {
    "material_id": "mp-124",
    "formula": "Ag",
    "composition": {
      "Ag": 1
    },
    "is_stable": false,
    "energy_above_hull": 0.0021267149999990003,
    "search_mode": "composition_only",
    "search_criteria": {
      "element": "Ag",
      "formula": null,
      "chemsys": null,
      "spacegroup_number": null,
      "crystal_system": null,
      "theoretical": false
    },
    "num_matches_found": 1299,
    "precalculated_bulk_moduli": [
      {
        "element": "AG",
        "name": "Silver",
        "bulk_modulus_GPa": [
          96,
          106
        ]
      }
    ],
    "bulk_modulus": {
      "k_vrh": 87.692,
      "k_voigt": 87.692,
      "k_reuss": 87.692,
      "unit": "GPa"
    },
    "shear_modulus": {
      "g_vrh": 13.21,
      "g_voigt": 14.24,
      "g_reuss": 12.179,
      "unit": "GPa"
```

```
      },
      "derived": {
        "poisson_from_KG": 0.42828083942002126,
        "youngs_from_KG": 37.73517977747697,
        "pugh_K_over_G": 6.6383043149129435
      },
      "mechanical_stability": {
        "likely_stable": true,
        "flags": null
      },
      "data_quality": "ok",
      "universal_anisotropy": 0.846,
      "poisson_ratio": 0.428
    },
    "has_image": false,
    "has_html": false,
    "notes": [
      "Structure parameters (spacegroup_number and crystal_system) not provided. Results may include
     multiple materials with different structures."
    ],
    "confidence": "high"
  }
}
```


Model Response

> The bulk modulus of silver (Ag) is approximately 87.692 GPa. This information has been obtained from the Materials Project database, known for its data reliability and accuracy in material properties.

A.1.4. Example 4: Doping Site Analysis (Semiconductors)

User Query

```
Does Fe prefer Ga or N sites in GaN?
```

Tool Call: analyze_doping_site_preference


```
{
  "host_formula": "GaN",
```

```
  "dopant_element": "Fe",
  "site_a_element": "Ga",
  "site_b_element": "N"
}
```


Tool Output


```
{
  "tool_name": "analyze_doping_site_preference",
  "result": {
    "success": true,
    "error": null,
    "error_type": null,
    "metadata": {
      "handler": "semiconductors",
      "function": "analyze_doping_site_preference",
      "timestamp": "2025-12-02T21:42:25.185064Z",
      "version": "0.0.1",
      "duration_ms": 23322.169303894043
    },
    "citations": [
      "Materials Project",
      "pymatgen"
    ],
    "data": {
      "host_material": {
        "material_id": "mp-804",
        "formula": "GaN",
        "energy_per_atom": -10.9558127175,
        "unit": "eV/atom"
      },
      "dopant_element": "Fe",
      "site_a": "Ga",
      "site_b": "N",
      "temperature_K": 298.15,
      "pressure_atm": 1,
      "site_a_substitution": {
        "material_id": "mp-3210158",
        "formula": "Nb12Ga9Fe3N4",
        "composition": {
          "Nb": 12,
          "Ga": 9,
          "Fe": 3,
          "N": 4
        },
        "energy_per_atom": -17.73532480214286,
        "energy_above_hull": 0.007623802244896001,
        "is_stable": false,
```

```
        "inferred_substitution_site": "Ga"
      },
      "site_b_substitution": null,
      "site_preference": {
        "preferred_site": "Ga",
        "energy_difference": null,
        "unit": null,
        "interpretation": "Fe is chemically closer to Ga (valence/electronegativity match). Score
    margin ~2.60 (lower score = better match).",
        "method": "chem-heuristic-fallback"
      },
      "note": "Only one doped sublattice had candidate structures with energies; fell back to valence/
    electronegativity heuristic for final comparison."
    },
    "has_image": false,
    "has_html": false,
    "notes": [
      "Site preference determined by comparing total energies"
    ],
    "confidence": "medium",
    "caveats": [
      "Assumes dilute doping limit",
      "Does not account for defect-defect interactions"
    ]
  }
}
```


Model Response

> Iron (Fe) shows a preference for substituting Ga sites in GaN. This conclusion is based on a heuristic considering valence and electronegativity match, with Fe being chemically closer to Ga. The preference was confirmed by analyzing candidate structures, and Fe aligned more closely with Ga in these analyses [Materials Project, pymatgen].

A.1.5. Example 5: Phase Diagram Generation (CALPHAD)

User Query

```
Plot the Al-Zn binary phase diagram
```

Tool Call: plot_binary_phase_diagram

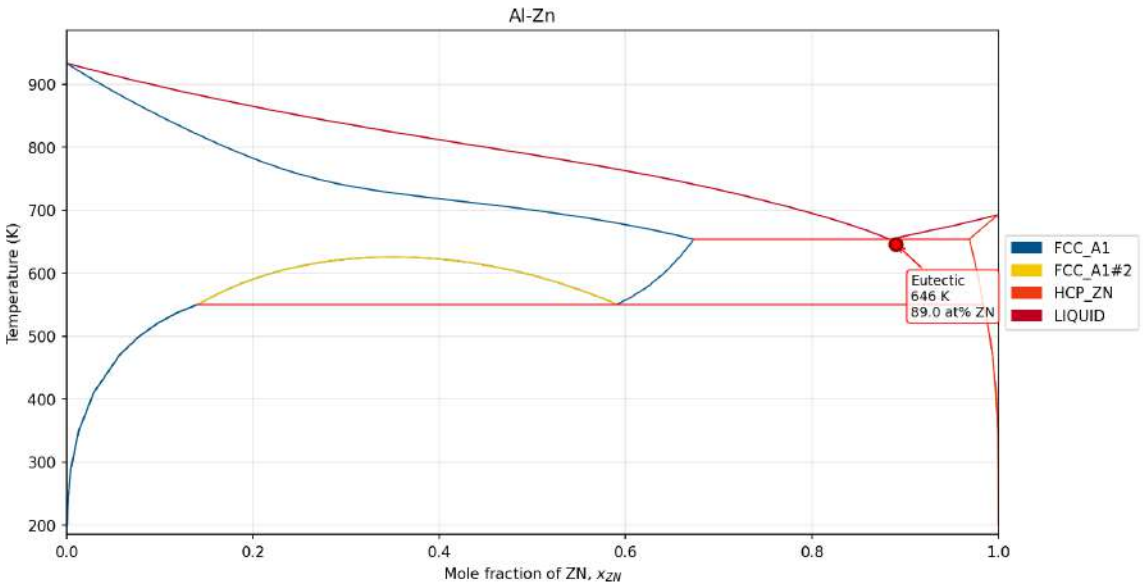

```
{
  "system": "Al-Zn"
}
```


Tool Response


```
{
  "tool_name": "plot_binary_phase_diagram",
  "result": {
    "success": true,
    "error": null,
    "error_type": null,
    "metadata": {
      "handler": "calphad",
      "function": "plot_binary_phase_diagram",
      "timestamp": "2025-12-02T21:44:15.133849Z",
      "version": "0.0.1",
      "duration_ms": 14525.683879852295
    },
    "citations": [
      "pycalphad"
    ],
    "data": {
      "message": "Successfully generated AL-ZN phase diagram showing phases: LIQUID, FCC_A1, HCP_ZN.
     Temperature range: 185-986 K. Pure AL melting point: 922 K. Pure ZN melting point: 686 K.
     Eutectic point: 646 K at 89.0 at% ZN (L → HCP_ZN + FCC_A1).",
      "system": "AL-ZN",
      "phases": [
        "LIQUID",
        "FCC_A1",
        "HCP_ZN"
      ],
      "temperature_range_K": [
        184.59713600881742,
        985.546063550312
      ],
      "key_points": [
        {
          "type": "pure_melting",
          "element": "AL",
          "temperature": 921.875
        },
        {
          "type": "pure_melting",
          "element": "ZN",
```

```
          "temperature": 685.625
        },
        {
          "type": "eutectic",
          "xB": 0.8900000000000081,
          "temperature": 646.2500000000437,
          "composition_pct": 89.00000000000081,
          "solids": [
            "HCP_ZN",
            "FCC_A1"
          ],
          "reaction": "L → HCP_ZN + FCC_A1",
          "validation": {
            "dT": 10,
            "dx": 0.03
          }
        }
      ]
    },
    "has_image": true,
    "image_url": "http://localhost:8000/static/plots/phase_diagram_AL-ZN_1764711854844.png",
    "has_html": false,
    "confidence": "high"
  }
}
```


Generated Visualization



Model Response

> The Al-Zn binary phase diagram has been successfully generated, showing the following key points and phases:
> Phases: LIQUID, FCC_A1, HCP_ZN Temperature Range: 185 - 986 K Pure Aluminum (Al) Melting Point: 922 K Pure Zinc (Zn) Melting Point: 686 K Eutectic Point: 646 K at 89.0 at% Zn You can view the phase diagram here: Al-Zn Phase Diagram
> This data comes from the CALPHAD method, as cited: pycalphad.

A.2. Example Drop-4 Claim Trace: Silver Bulk Modulus Feasibility Claim

Figures 2.1 and 2.2 show a sample LAMINA interaction where the system evaluates the claim *"The bulk modulus of Silver is* ~ 100 *GPa."* For completeness, we reproduce the full text prompt, model response, and tool call JSON used in that interaction.

User Query

```
Evaluate the following materials science claim and provide a
feasibility rating.

Claim: The bulk modulus of Silver is ~100 GPa

Rate the claim's feasibility on a scale from -2 to +2:

- -2: Extremely infeasible (clearly contradicted by evidence)
- -1: Likely infeasible (evidence suggests this is unlikely)
-  0: Undecidable (insufficient evidence or vague)
- +1: Likely feasible (evidence suggests this is plausible)
- +2: Extremely feasible (strongly supported by evidence)

Use computational tools to verify the claim if appropriate.
Provide your reasoning, then end your response with exactly
this format on its own line:

VERDICT:
```

Tool Call: get_elastic_properties


```
{
  "element": "Ag"
}
```


Tool Response


```
{
  "tool_name": "get_elastic_properties",
  "result": {
    "success": true,
    "error": null,
    "error_type": null,
    "metadata": {
      "handler": "materials",
      "function": "get_elastic_properties",
      "timestamp": "2025-11-30T20:56:52.578508Z",
      "version": "0.0.1",
```

```
    "duration_ms": 838.1671905517578
  },
  "citations": [
    "Materials Project"
  ],
  "data": {
    "material_id": "mp-124",
    "formula": "Ag",
    "composition": {
      "Ag": 1
    },
    "is_stable": false,
    "energy_above_hull": 0.0021267149999990003,
    "search_mode": "composition_only",
    "search_criteria": {
      "element": "Ag",
      "formula": null,
      "chemsys": null,
      "spacegroup_number": null,
      "crystal_system": null,
      "theoretical": false
    },
    "num_matches_found": 1299,
    "precalculated_bulk_moduli": [
      {
        "element": "AG",
        "name": "Silver",
        "bulk_modulus_GPa": [
          96,
          106
        ]
      }
    ],
    "bulk_modulus": {
      "k_vrh": 87.692,
      "k_voigt": 87.692,
      "k_reuss": 87.692,
      "unit": "GPa"
    },
    "shear_modulus": {
      "g_vrh": 13.21,
      "g_voigt": 14.24,
      "g_reuss": 12.179,
      "unit": "GPa"
    },
    "derived": {
      "poisson_from_KG": 0.42828083942002126,
      "youngs_from_KG": 37.73517977747697,
      "pugh_K_over_G": 6.6383043149129435
    },
    "mechanical_stability": {
      "likely_stable": true,
```

```
        "flags": null
      },
      "data_quality": "ok",
      "universal_anisotropy": 0.846,
      "poisson_ratio": 0.428
    },
    "has_image": false,
    "has_html": false,
    "notes": [
          "Structure parameters (spacegroup_number
          and crystal_system) not provided. Results
          may include multiple materials with
          different structures."
    ],
    "confidence": "high"
  }
}
```


A.2.1. Model Response

Model Response

[Model: gpt-4o-mini]

To evaluate the claim regarding the bulk modulus of silver being approximately 100 GPa, I retrieved the relevant properties from the Materials Project database. The results indicate a range of bulk modulus values for silver (Ag):

The bulk modulus of silver is reported to be between 96 GPa and 106 GPa.

Given this information, the claim that the bulk modulus of silver is 100 GPa is strongly supported by the evidence available. The values are very close to the stated claim, falling well within the range provided by computational results.

Based on this analysis, I would rate the claim's feasibility as follows:

VERDICT: +2

A.3. Tool Registry and Function Reference

This appendix lists all tools currently available to LAMINA, grouped by subsystem. Each entry gives the function name and a brief description of its purpose and typical inputs. These are the functions Kani may call when evaluating materials science claims.

A.3.1. Thermodynamics (CALPHAD; 9 functions)

`plot_binary_phase_diagram` Preferred for phase-diagram questions. Generates a binary phase diagram for a chemical system (e.g., "Al-Zn", "aluminum-zinc") using CALPHAD data. Shows the full composition range with liquidus/solidus and phase boundaries; optional control over temperature range, composition step, figure size, and pressure.

`plot_composition_temperature` Preferred for composition-specific thermodynamic questions. Plots phase stability vs. temperature for a specific composition (e.g., "Al20-Zn80", "Al80Zn20" or a pure element like "Zn"). Shows which phases are stable at different temperatures. Supports atomic vs. weight percent, configurable temperature range, and interactive Plotly output (HTML).

`analyze_last_generated_plot` Analyzes and interprets the most recently generated phase diagram or composition–temperature plot. Provides text commentary on phase boundaries, invariant reactions, melting points, and other thermodynamic insights based on the actual generated plot.

`calculate_equilibrium_at_point` Calculates equilibrium phase fractions at a single composition and temperature. Inputs are composition (e.g., "Al30Si55C15","Al80Zn20"), temperature in Kelvin, composition units (atomic or weight percent), and optional pressure. Returns detailed phase information including phase fractions and

phase compositions.

`calculate_phase_fractions_vs_temperature` Computes phase fractions as a function of temperature for a fixed composition. Used for studying precipitation, dissolution, and phase transformations. Returns phase-fraction curves over a specified temperature range (with configurable step size and composition units).

`analyze_phase_fraction_trend` Analyzes how the fraction of a specific phase (e.g., `AL4C3`, `SiC`, `FCC_A1`) changes with temperature between two bounds. Summarizes precipitation/dissolution trends and can optionally include Scheil–Gulliver solidification predictions for as-cast microstructures.

`verify_phase_formation_across_composition` Verifies phase-formation statements across a composition range. Used for claims like "beyond X% of element B, phase Y forms" or "above 50 at.% Al, phase Z appears". For a given system (binary or ternary), phase name/category, and composition threshold, it checks which phases are present at a specified temperature and reports whether the claim is supported.

`sweep_microstructure_claim_over_region` Sweeps a composition region and evaluates microstructure properties over a grid of alloys. Used to test claims of the form "all Al-Mg-Zn alloys with Mg < 8 at.% and Zn < 4 at.% form fcc+tau microstructures" or "phase fraction of τ is always $> 20\%$". Takes a chemical system, element ranges, claim type (two-phase, three-phase, phase-fraction), and returns a grid of CALPHAD results indicating where the claim holds or fails.

`fact_check_microstructure_claim` Evaluates specific microstructure claims for a given multicomponent alloy: e.g., "Al-8Mg-4Zn forms fcc+τ with $\tau < 20\%$" or "the eutectic has $> 20\%$ intermetallics". Takes a system, composition, claim type, expected phases and/or phase fractions, and a process model (as-cast vs. equilibrium at

a specified temperature). Returns a score on a $[-2, 2]$ scale plus supporting thermodynamic data. Mainly used as a helper specifically for Drop-4.

A.3.2. Materials Project / DFT / CHGNet (8 functions)

`mp_search_by_composition` Searches materials by chemical system, formula, or elements using the Materials Project API. At least one of `chemsys`, `formula`, or `element` must be provided (e.g., "Li-Fe-O", "Fe2O3", or "Li,Fe,O"). Returns matching materials with basic metadata.

`mp_get_by_id` Fetches materials by their Materials Project IDs (e.g., `mp-149`, `mp-30`). Supports a configurable list of fields, including structural, energetic, electronic, magnetic, dielectric, surface, and metadata fields. Used when the model already has specific MP IDs to inspect.

`mp_get_by_characteristic` Searches materials by property ranges and descriptors (band gap, density, elastic moduli, dielectric constants, energy above hull, magnetization, etc.), along with elements, crystal system, stability flags, and more. Useful for queries like "find cubic oxides with band gap between 1.2 and 3 eV and bulk modulus > 100 GPa".

`mp_get_material_details` Fetches one or more MP materials and returns detailed information similar to `mp_get_by_id`, with an option to return all fields. Used when a claim references specific MP entries and detailed structural/energetic context is needed.

`get_elastic_properties` Retrieves elastic and mechanical properties for a material. Supports two modes: (i) by material ID, or (ii) by composition (element/formula/chemsys) plus space group and crystal system. Returns bulk/shear moduli (Voigt,

Reuss, VRH), Poisson ratio, derived Young's modulus, Pugh ratio, anisotropy, and mechanical stability flags, optionally recomputing from the elastic tensor when available.

`find_closest_alloy_compositions` Finds alloy entries with compositions closest to a specified target (e.g., Ag–Cu alloys near 12.5% Cu). Takes a list of elements, an optional target composition dict, and stability/energy-above-hull filters. Returns stable (or mildly metastable) binaries or multicomponent alloys near the target composition.

`compare_material_properties` Compares a specific property between two MP materials, given their IDs and a property name (e.g., bulk modulus, shear modulus, Poisson ratio). Returns a side-by-side comparison to support comparative claims.

`analyze_doping_effect` Analyzes the effect of doping a host material with a dopant element on a selected property (default: bulk modulus). Takes a host element, dopant element, dopant fraction, and property name, and uses MP data and VRH-type averages to estimate how the property changes with doping.

A.3.3. Electrochemistry (8 functions)

`search_battery_electrodes` Searches for battery electrode materials and their voltage profiles for a given working ion (Li, Na, Mg). Inputs can be formula, element list, and optional constraints on capacity and average voltage. Returns candidate electrodes with their electrochemical performance.

`calculate_voltage_from_formation_energy` Calculates electrode voltage using MP DFT formation energies via convex-hull (InsertionElectrode) constructions. Takes the electrode formula and working ion (Li/Na/Mg) and returns voltage information based on a consistent MP entry set (nominally at 0 K).

`get_voltage_profile` Returns a detailed voltage profile and phase-evolution data for a specific electrode, given its MP battery/material ID and working ion. Used to visualize charge/discharge curves and plateau behavior.

`compare_electrode_materials` Compares multiple electrode materials (e.g., "Al,AlMg" or "$LiCoO_2$,$LiFePO_4$") for a given working ion. Returns average voltages, capacities, and energy densities to answer questions like "does alloying increase or decrease the voltage vs. Li/Li^+?".

`check_composition_stability` Checks whether a composition is thermodynamically stable (on the convex hull) and optionally whether it is stable as a function of voltage (grand-potential phase diagrams). Used to answer "can this composition form as a stable phase?" or "is it stable as a Li anode?". Returns energy above hull, decomposition products, and optionally voltage-dependent behavior.

`analyze_anode_viability` Analyzes whether a given composition can function as a battery anode for a given working ion (default: Li). Combines thermodynamic stability checks and voltage calculations over a specified voltage window, and returns a verdict on anode viability plus supporting data.

`analyze_lithiation_mechanism` Analyzes the lithiation mechanism of a host material (Al, Mg, AlCu, etc.) without including Li in the input formula. Reports phase evolution, two-phase vs. single-phase reactions, and equilibrium phases at different Li contents. Includes strict 0 K classification and effective two-phase classification for near-degenerate multi-phase plateaus.

`estimate_ion_hopping_barrier` Uses CHGNet NEB to estimate ion hopping barriers in graphite/graphene galleries only (bilayer graphene, stage-I and stage-II graphite). Enumerates realistic configurations (e.g., LiC_6, LiC_{12}, $\sqrt{3} \times \sqrt{3}$ Li ordering) and

returns ranked scenarios with median activation barriers, uncertainty bands, and Arrhenius kinetics at specified temperatures. Restricted to graphite/graphene hosts.

A.3.4. Magnetism / Defects / Alloys (11 functions)

`assess_phase_strength_and_stiffness_claims` Assesses claims about alloy microstructure, strengthening, embrittlement, and stiffness. For a given system (e.g., Fe–Al, Ni–Al, Al–Mg or ternary like Al–Mg–Zn), composition, and temperature, it (1) computes CALPHAD equilibrium phases, (2) identifies matrix vs. secondary phases, (3) estimates mechanical properties using MP elastic data, (4) decides whether secondary phases likely strengthen or embrittle, (5) estimates matrix stiffness via a rule-of-mixtures, and (6) verifies claims about phase formation and mechanical/stiffness changes.

`estimate_surface_diffusion_barrier` Estimates surface diffusion barriers using DFT + NEB or CHGNet NEB across multiple realistic scenarios: different facets ((111), (100), (110)), terrace vs. step-edge vs. vacancy-mediated paths, and variable adatom coverage. Returns ranked scenarios with P10/P50/P90 barrier estimates, kinetic rates, and diffusion coefficients at a specified temperature. Intended for quantitative diffusion predictions on metal surfaces.

`assess_doping_effect_on_saturation_mag...` Assesses how doping changes saturation magnetization (M_s) for a host material (e.g., Fe_2O_3, Co_3O_4). Takes a host formula, dopant element, and dopant fraction, and compares MP-derived M_s values for host vs. doped materials. Returns a quantitative analysis of whether M_s increases, decreases, or is roughly retained.

`assess_magnet_strength_with_doping` Evaluates whether doping makes a material a stronger permanent magnet. Considers pull force, coercivity, and magnetic ordering

to decide if the doped magnet "pulls better" than the baseline, given a host formula, dopant, dopant fraction, and magnet geometry (diameter, length, air gap).

`calculate_magnet_pull_force` Calculates the pull force (in N) for a cylindrical permanent magnet contacting steel, given remanence B_r and geometry (diameter, length, air gap, geometry factor). Used for translating B_r into practical magnet force comparisons.

`compare_dopants_for_saturation_magnetization` Compares multiple dopant elements (e.g., Ni, Co, Nd) to see which causes the least degradation or the greatest enhancement in M_s for a given host. Tests each dopant at a fixed fraction, and ranks them by M_s retention or improvement.

`estimate_permanent_magnet_properties` Estimates permanent magnet properties (saturation B_s, remanence B_r, coercivity H_c, and $(BH)_{\text{max}}$) from MP DFT magnetization data for a given formula. Uses a remanence factor (B_r/B_s, default 0.7 for hard magnets) and heuristic coercivity estimates.

`get_phase_and_magnetic_ordering` Returns phase information and magnetic ordering for a material with a given formula (e.g., Fe_2O_3). Includes crystal structure, space group, stability indicators, and magnetic ordering type (ferromagnetic, antiferromagnetic, ferrimagnetic, etc.).

`get_saturation_magnetization_detailed` Retrieves detailed M_s information for a material: saturation magnetization in Tesla, A/m, and emu/g, plus ordering type and site-level magnetic information. Useful for converting between units and for more nuanced magnetic comparisons.

`search_doped_magnetic_materials` Searches for doped versions of a host material by specifying host elements and a dopant. Returns MP materials containing all speci-

fied elements, sorted by stability and structural similarity, to provide concrete doped candidates for further M_s or magnetism analysis.

`evaluate_dopant_effect_on_Ms` For a given host oxide/semiconductor and a dopant, finds same-phase doped variants and computes how M_s changes in kA/m, only trusting cases where the doped cell has the same space group as the host. This tool is deliberately conservative and refuses to claim M_s improvement if the crystal phase changes.

A.3.5. Semiconductors (8 functions)

`analyze_octahedral_distortion_in_material` Analyzes octahedral distortions in a crystal structure, given an MP material ID and a central element (e.g., V in VO_2) and optionally a neighbor element (e.g., O). Computes bond lengths and distortions in the octahedral coordination environment, useful for understanding structural transitions.

`analyze_defect_stability` Analyzes defect formation energy for substitutional or interstitial dopants. Requires a host material ID (e.g., `mp-149` for Si) and an explicit doped-supercell composition (e.g., {`Si`:31, `P`:1} for one P substituting Si in a 32-atom cell). Compares substitutional vs. interstitial configurations and can be used to rank defect stabilities.

`analyze_doping_site_preference` Analyzes doping site preference in compound semiconductors such as GaAs. Given a host formula, dopant element, and two candidate sublattice elements (e.g., Ga vs. As), it compares energies for placing the dopant on each site at a specified temperature and pressure, and reports which site is preferred.

`analyze_phase_transition_structures` Analyzes temperature-dependent structural changes by comparing different polymorphs of a given formula (e.g., VO_2). Can

focus on a specific element and neighbor species (e.g., V–O) to track coordination changes across polymorphs, supporting qualitative phase-transition reasoning.

compare_magnetic_materials Compares magnetic properties between two materials (by MP ID), such as undoped vs. doped semiconductors. Used to assess whether doping enhances or suppresses magnetization.

get_magnetic_properties Retrieves detailed magnetic properties for a given MP material ID, including total magnetization, ordering type, and site information. Often used together with `compare_magnetic_materials`.

predict_defect_site_preference Predicts whether a dopant prefers substitutional or interstitial sites in a semiconductor host, using physics-based heuristics by default, or explicit DFT formation energies if provided. Can optionally take an MP ID for structural context and formation energies for substitutional and tetrahedral/octahedral interstitial configurations.

search_same_phase_doped_variants Searches for plausible substitutional doped variants of a host phase without changing the overall structure type. For example, finds Fe_2O_3-like phases with a few percent Co on the Fe sublattice, but avoids completely different compounds (e.g., spinels). Filters candidates by crystal system, space group, stoichiometry pattern, and dopant fraction. Intended as a pre-filter; magnetic claims must be validated with the dedicated M_s tools.

A.3.6. Superconductors (1 function)

analyze_cuprate_octahedral_stability Analyzes how c-axis spacing affects Cu–O octahedral stability in cuprate superconductors, given a formula (e.g., La_2CuO_4, $YBa_2Cu_3O_7$). The tool is family-aware (214-type, 123-type, T′, infinite-layer) and

can reason about hypothetical increases/decreases in c or about observed c values. Returns trends in octahedral stability and Jahn–Teller distortions under different scenarios.

A.3.7. Solutes (4 functions)

`analyze_solute_lattice_effect` Preferred tool for lattice-parameter analysis in fcc matrices. Given a matrix element (e.g., Al), solute (e.g., Mg), solute concentration (at.%), and temperature, it: (i) uses CALPHAD to validate that the solute is actually soluble in the matrix at the specified conditions, and (ii) combines Vegard's law and size-misfit theory to predict whether the lattice expands or contracts and by how much. Includes Hume–Rothery reasoning.

`calculate_solute_lattice_effect` A lower-level interface that assumes CALPHAD equilibrium data is already known. Takes matrix/solute elements, solute concentration, temperature, matrix phase name (must be `FCC_A1`), and matrix-phase composition from CALPHAD, then computes the lattice-parameter change. Intended for advanced use; most queries should go through `analyze_solute_lattice_effect`.

`compare_solute_lattice_effects` Preferred tool for comparing multiple solutes. Given a matrix element (e.g., Al) and a list of solutes (e.g., Mg, Cu, Zn) at a fixed nominal concentration, it automatically runs CALPHAD for each solute, validates solubility, computes $\Delta a/a$, and returns a ranking from largest expander to largest contractor.

`get_solute_reference_data` Returns reference data for fcc lattice parameters and metallic radii, and lists which matrix–solute combinations are supported by the solute-analysis tools. Useful to check whether a proposed system is covered.

A.3.8. Search (1 function)

`search_web` Searches the web and scientific literature using a SearXNG frontend. Supports: (i) general web search, (ii) scientific literature search (with optional arXiv, PubMed, and Scholar), and (iii) materials-specific search emphasizing phase diagrams and thermodynamic data. Options include language, time range, pagination, selective content extraction, and image extraction. Used when questions involve materials or conditions not well-covered by the in-house CALPHAD or MP toolsets.

A.4. Example Failure Traces

A.4.1. Routing/Selection Error

Claim: "At low temperatures there are no distortions in the VO6 octahedra in VO2"

Gold Label: -2 (infeasible)

Tool Trace:

1. `search_web(query="VO2 octahedral distortion low temperature")` $\rightarrow$ Returns irrelevant results

2. `analyze_octahedral_distortion_in_material(material_id="mp-10215 22",...)` $\rightarrow$ Returns distortion data (correct tool, eventually called)

Issue: The router first selects web search (a generic fallback) wasting a tool call on an irrelevant result before invoking the specialized semiconductor analysis tool. While the final prediction (-2) is correct, the routing inefficiency shows how policy gaps can cause unnecessary latency and noise.

A.4.2. Parameterization Error from Vague Quantifier

Claim: "In the Al-Mg-Zn system, adding just a few atomic percent Zn to the eutectic Al-Mg composition with about 34.5 at.% Mg leads to a significant amount of the tau phase: over 20% phase fraction."

Gold Label: $+2$ (feasible)

Tool Trace (across H3 mappings):

- m_1 ("few" $\mapsto$ 2% Zn): `calculate_equilibrium` $\rightarrow$ tau phase = 12.1%
  $\Rightarrow$ predicted -2
- m_2 ("few" $\mapsto$ 3% Zn): `calculate_equilibrium` $\rightarrow$ tau phase = 21.9%
  $\Rightarrow$ predicted $+2$
- m_3 ("few" $\mapsto$ 5% Zn): `calculate_equilibrium` $\rightarrow$ tau phase = 41.5%
  $\Rightarrow$ predicted $+2$

Issue: The phrase "a few atomic percent" is numerically vague. The CALPHAD tool executes correctly for each interpretation, but the 2% $\rightarrow$ 3% boundary crosses the 20% tau-phase threshold, flipping the verdict from infeasible to strongly supported.

A.4.3. Verbalization Drift

Claim: "In binary iron-aluminum ferroalloys, concentrations of aluminum of about 65 at.% lead to small amounts of a secondary intermetallic Al2Fe phase which increases mechanical strength of the *primary BCC iron phase* without causing embrittlement."

Gold Label: -2 (infeasible)

Tool Output (excerpt):

```
phases: [
  {name: "AL2FE", fraction: 0.944},
  {name: "BCC_A2", fraction: 0.056}
]
```

Model Reasoning: "The presence of the Al2Fe phase is associated with high likelihood of strengthening... The yield strength is estimated to be 475.6 MPa... embrittlement risk is low."

Predicted Label: $+2$ (strongly supported)

Issue: The claim explicitly states that BCC iron is the *primary* phase with "small amounts" of Al2Fe as secondary. The tool output directly contradicts this: Al2Fe is 94.4% of the microstructure (the primary phase), while BCC is only 5.6%. The model correctly reports the strengthening metrics but fails to notice that a fundamental premise of the claim (which phase is primary) is violated. This verbalization drift results in a $+2$ verdict when the claim is structurally infeasible (-2).

A.5. Drop-4 Claims: Precise vs. Vague

A.5.1. Precise Claims

Table A.1: Precise Drop-4 claims, grouped by domain and claim type. Full verbatim claim text appears below.

Idx	Claim	Gold	Type
Thermodynamics / phase behavior: 7 claims			
1	Al20Zn80 at 870K is solid at equilibrium	-2	Explicit state point
2	Al50Zn50 forms a single solid phase below 700K	0	Explicit threshold
3	Al30Si55C15 carbide fraction increases as temperature decreases	-2	Explicit monotone trend
4	Al30Si55C15 retains carbide precipitates even at 1500K	1	Explicit dissolution claim
11	Fe30Al70 at 750K gives $Al_{13}Fe_4$ + Al-rich BCC Fe	-2	Explicit phase assemblage
12	Fe30Al70 at 750K gives Al_5Fe_2 + Al_2Fe	2	Explicit phase assemblage
13	Fe30Al70 at 750K gives 5% Al_5Fe_2 + 95% Al_2Fe	-2	Explicit phase fractions
Electrochemistry: 3 claims			
5	Mg alloying in Al anodes increases Li-battery voltage	0	Voltage change claim
6	Cu80Li10Al10 forms a thermodynamically stable Li-battery anode	-2	Stability claim
9	Li hopping barrier in graphite is below 0.1 eV	1	Explicit barrier bound
Elastic / mechanical / lattice: 3 claims			
7	12.5% Cu in Ag increases bulk modulus by 20%	-2	Explicit modulus change
8	Au diffusion barrier on Al surfaces is below 0.2 eV	-1	Explicit barrier bound
15	Mg defects increase Al fcc lattice parameter more than Cu or Zn	2	Explicit ranking
Defects / magnetism: 2 claims			

Continued on next page

Idx	Claim	Gold	Type
10	In GaAs at STP, N on Ga sites is more stable than N on As sites	-2	Site-preference ordering
14	Co doping raises Fe_2O_3 saturation magnetization by up to 10%	-2	Explicit Ms bound

1. Al20Zn80 at 870K is a solid at equilibrium.
2. Al50Zn50 forms a single solid phase $<$ 700K.
3. The phase fraction of Aluminum Silicon Carbide precipitates in Al30Si55C15 increases with decreasing temperature from 500K to 300K.
4. Heating a sample of Al30Si55C15 to 1500K is insufficient to dissolve all carbide precipitates.
5. Alloying Mg into an Al anode increases the voltage of a lithium battery.
6. Cu80Li10Al10 can form a thermodynamically stable anode for a lithium battery.
7. Doping Ag with 12.5% Cu increases the bulk modulus by 20%.
8. Gold diffusion barrier on aluminum surfaces is < 0.2eV.
9. The barrier for Li hopping between sites in a graphite electrode is < 0.1eV.
10. At STP, nitrogen doping of GaAs at Ga sites is more stable than at As sites.
11. A mixture of 30 at.% Fe and 70 at.% Al at 750K results in a two-phase solid mixture of Al13Fe4 and Al-rich BCC Fe.
12. A mixture of 30 at.% Fe and 70 at.% Al at 750K results in a two-phase solid mixture of Al5Fe2 and Al2Fe.
13. A mixture of 30 at.% Fe and 70 at.% Al at 750K results in a two-phase mixture with 5% Al5Fe2 and 95% Al2Fe.

14. The saturation magnetization of hematite iron oxide (Fe_2O_3) can be increased by up to 10% by doping cobalt onto the iron sites.

15. Among Cu, Mg, and Zn additives in aluminum alloys, magnesium substitutional defects cause the largest increase in the equilibrium lattice parameter of the primary aluminum fcc phase.

A.5.2. Vague Claims and Mapping Types

Table A.2: Vague Drop-4 claims, grouped by ambiguity family. Full mapping details and verbatim claim text are provided in the benchmark release, and the full verbatim claim text appears below.

Idx	Claim	Gold	# Maps	Mapping family
Approximate numeric thresholds / windows: 11 claims				
1	Al–Zn eutectic near Al15Zn75	2	3	Approximate composition window
2	AlMg anode OCP is about 0.5 V vs. Li/Li^+	-2	3	Approximate voltage window
5	Ag bulk modulus is about 100 GPa	2	3	Approximate modulus window
10	In Fe–Al, beyond about 50 at.% Al forms secondary Al_2Fe	2	3	Approximate composition threshold
15	A few at.% Zn in eutectic Al–Mg yields $>20\%$ τ	2	4	Vague quantity
17	Fe doping significantly increases Co_3O_4 saturation magnetization	2	4	Significance threshold
18	Nd doping significantly increases Co_3O_4 saturation magnetization	-1	4	Significance threshold
19	Soluble Cu significantly increases elastic modulus of primary Al	2	3	Significance threshold
20	Soluble Cu significantly decreases elastic modulus of primary Al	-2	3	Significance threshold

Continued on next page

Idx	Claim	Gold	# Maps	Mapping family
21	Soluble Mg significantly increases elastic modulus of primary Al	1	3	Significance threshold
22	Cu defects moderately increase Al fcc lattice parameter	-1	5	Magnitude threshold
Semantic target / criterion ambiguity: 4 claims				
3	Practical maximum capacity for $(CuAl)_{1-x}Li_x$ is at $x = 0.4$	-1	1	Criterion definition
7	Increasing c-axis spacing stabilizes Cu–O octahedra in La_2CuO_4	-1	2	Stability definition
8	Al-doped α-Fe_2O_3 makes a stronger magnet	2	4	Target metric definition
16	In Fe_2O_3, Co causes the least degradation in M_s among dopants	1	1	Relative criterion definition
Process / condition ambiguity: 5 claims				
4	Initial lithiation of AlCu is a two-phase reaction	2	2	Mechanism definition
6	At low temperature, VO_2 has no VO_6 octahedral distortions	-2	3	Temperature-range ambiguity
9	P interstitials in Si are unstable relative to substitutional P	2	2	Chemical-potential condition
13	Mostly-Al with Mg $<$8 at.% and Zn $<$4 at.% gives desirable fcc + τ	2	3	Process + desirability definition
14	Al–8Mg–4Zn gives desirable fcc + Laves MgZn + γ-AlMg	-1	3	Process + desirability definition
Compound design ambiguity: 2 claims				

Continued on next page

Idx	Claim	Gold	# Maps	Mapping family
11	Fe–Al near 50–52 at.% Al: small Al_2Fe strengthens without embrittlement	1	3	Multi-criterion design claim
12	Fe–Al near 65 at.% Al: small Al_2Fe strengthens BCC Fe without embrittlement	-2	3	Multi-criterion design claim

1. Al and Zn form an eutectic at ∼Al15Zn75.
2. The OCP for a lithium cell with an AlMg anode is ~ 0.5 V vs. Li/Li^+.
3. For a $(CuAl)_{1-x}Li_x$ electrode, the maximum practical capacity is $x = 0.4$.
4. The initial reaction of Li with AlCu is a two phase reaction.
5. The bulk modulus of silver is ~ 100 GPa.
6. At low temperatures there are no distortions in the VO_6 octahedra in VO_2.
7. Increasing spacing along the c-axis stabilizes the Cu–O octahedra in La_2CuO_4.
8. Stronger magnet made by doping α-Fe_2O_3 with Al.
9. Phosphorus interstitials in Si are unstable compared to substitutional doping.
10. In Fe–Al ferroalloys, increasing Al beyond about 50 at.% forms secondary Al_2Fe.
11. Fe–Al with about 50–52 at.% Al: small Al_2Fe fraction increases strength without embrittlement.
12. Fe–Al with about 65 at.% Al: small Al_2Fe increases strength of primary BCC Fe without embrittlement.

13. Mostly Al with $<$8 at.% Mg and $<$4 at.% Zn: slow solidification yields mechanically desirable fcc + τ (up to 20%).

14. Al–8Mg–4Zn: slow solidification yields mechanically desirable fcc + Laves MgZn + γ-AlMg.

15. Adding just a few at.% Zn to eutectic Al–Mg (34.5 at.% Mg) yields $>$20% τ phase.

16. For Fe_2O_3, Co causes the least degradation in saturation magnetization among Ni, Co, Nd dopants.

17. Co_3O_4 saturation magnetization can be significantly increased by Fe doping.

18. Co_3O_4 saturation magnetization can be significantly increased by Nd doping.

19. Soluble Cu in the primary Al phase significantly increases elastic modulus.

20. Soluble Cu in the primary Al phase significantly decreases elastic modulus.

21. Soluble Mg in the primary Al phase significantly increases elastic modulus.

22. Cu substitutional defects cause a moderate increase in the Al fcc lattice parameter.

APPENDIX B : PathFinder Supplementary Materials

B.1. Complete Tool Registry

Table B.1: Complete tool registry (44 tools). During evaluation, all catalog discovery tools (9), strategy mutation (5), execution (1), sample results (1 of 2), inspection (1), and research (2) are enabled (19 tools total, Table 4.1). The remaining 25 tools support production features not relevant to strategy construction.

Category	**Tools**	**#**
Catalog discovery	`get_record_types`, `search_for_searches`, `browse_search_categories`, `list_searches`, `list_transforms`, `get_search_parameters`, `get_dependent_vocab`, `lookup_phyletic_codes`, `search_example_plans`	9
Strategy mutation	`create_step`, `create_colocation_step`, `update_step`, `delete_step`, `undo_last_change`	5
Step attachments	`add_step_filter`, `add_step_analysis`, `add_step_report`	3
Execution	`get_estimated_size`	1
Results	`get_sample_records`, `get_download_url`	2
Inspection	`get_strategy`	1
Conversation	`rename_strategy`, `clear_strategy`	2
Research	`web_search`, `literature_search`	2
Gene resolution	`lookup_gene_records`, `resolve_gene_ids_to_records`	2
Planning & analysis	`save_planning_artifact`, `set_conversation_title`, `report_reasoning`, `optimize_search_parameters`, `run_control_tests_on_step`, `run_control_tests_on_search`, `export_gene_set`	7

Continued on next page

Category	Tools	#
Workbench	`create_workbench_gene_set`, `run_gene_set_enrichment`, `list_workbench_gene_sets`, `get_evaluation_summary`, `get_enrichment_results`, `get_confidence_scores`, `get_step_contributions`, `get_experiment_config`, `get_ensemble_analysis`, `get_result_gene_lists`	10
Total		**44**

B.2. Gold-Standard Strategy Corpus

Table B.2: The 35 gold-standard strategies, grouped by complexity bin and sorted by AST node count within each bin. †Public user strategies from VEuPathDB's shared repository. Database coverage spans 8 sites: PlasmoDB (12), FungiDB (7), ToxoDB (4), TriTrypDB (4), CryptoDB (3), VectorBase (3), GiardiaDB (1), MicrosporidiaDB (1).

Strategy	Database	Nodes	Source
Small (1–4 nodes): 9 strategies			
`toxo_oocyst_upregulated`	ToxoDB	1	Workshop
`afumigatus_upregulated_exons`	FungiDB	3	Publication
`agambiae_hemocyte_secreted`	VectorBase	3	Workshop
`calbicans_infection_specific`	FungiDB	3	Workshop
`pvivax_tm_transporter`	PlasmoDB	3	Workshop
`tcruz_active_trans_sialidase`	TriTrypDB	3	Publication
`text_alphafold_fungal`	FungiDB	3	Publication
`crypto_genes_orthologs`†	CryptoDB	4	Public strategy
`plasmo_liver_stage_scrnaseq`	PlasmoDB	4	Publication
Medium (5–8 nodes): 15 strategies			
`cneoformans_meningitis_expression`	FungiDB	5	Publication

Continued on next page

Strategy	Database	Nodes	Source
eliasalexander†	GiardiaDB	5	Public strategy
kinase_signalpep_phosphorylation	PlasmoDB	5	Workshop
pgraminis_effector_motif	FungiDB	5	Publication
protease_gametocyte	PlasmoDB	5	Training module
protein_coding_signal_peptide†	CryptoDB	5	Public strategy
kinase_pberghei_orthologs_expanded	PlasmoDB	6	Workshop
protease_ortholog_transform	PlasmoDB	6	Training module
toxo_apicoplast_orthologs	ToxoDB	6	Publication
kinase_transmembrane_expanded	PlasmoDB	7	Publication
anopheles_midgut_proteases	VectorBase	8	Publication
cali_workshop_2024†	VectorBase	8	Public strategy
kinase_tm_est_proteomics†	CryptoDB	8	Public strategy
protease_snp_colocation	PlasmoDB	8	Training module
rodríguez_duran_et_al†	TriTrypDB	8	Public strategy
Large (9+ nodes): 11 strategies			
microsporidia_control†	MicrosporidiaDB	9	Public strategy
rodríguez_duran_solo†	TriTrypDB	9	Public strategy
afd_vaccine_strategy†	PlasmoDB	10	Public strategy
exclusive_genes_l_infantum†	TriTrypDB	10	Public strategy
gct_genes_in_pb_pf†	PlasmoDB	10	Public strategy
magnaporte_grisea†	FungiDB	11	Public strategy
putative_backbone_nrps†	FungiDB	11	Public strategy
toxo_gama_secreted_invasion	ToxoDB	11	Publication
tg_drug_targets†	ToxoDB	12	Public strategy
plasmo_candidate_drug_targets	PlasmoDB	16	Publication

Continued on next page

Strategy	Database	Nodes	Source
possible_vaccine_candidate†	PlasmoDB	17	Public strategy

B.3. WDK Search Types Exercised By The Gold Corpus

Table B.3: WDK search types exercised by the 35 gold strategies. *Strategies* = number of gold strategies using this search type; *Variants* = number of dataset-specific concrete search names (e.g., the 13 RNA-Seq variants correspond to 13 organism-specific datasets). Types marked with * have dataset-specific suffixes in the WDK API. Total: 31 base types, 44 concrete search names.

Category	Base search type	Strategies	Variants
Text & annotation	GenesByText	16	1
	GenesByGoTerm	10	1
	GeneByLocusTag	1	1
Expression	GenesByRNASeq*	12	13
	GenesByMassSpec	6	1
	GenesByMicroarray*	3	2
	GenesByProteomics*	1	1
	GenesBySingleCell	1	1
	GenesByToxoProfileSimilarity	1	1
Protein features	GenesWithSignalPeptide	12	1
	GenesByTransmembraneDomains	10	1
	GenesByInterproDomain	2	1
	GenesByEcNumber	2	1
	GenesByMolecularWeight	2	1
	GenesByMotifSearch	2	1
	GenesWithAlphafold	1	1
	GenesByPdbSimilarity	1	1

Continued on next page

Category	Base search type	Strategies	Variants
Genomic features & variation	`GenesByNgsSnps`	3	1
	`GenesByESTOverlap`	3	1
	`GenesByGeneType`	2	1
	`GenesByExonCount`	1	1
	`GenesByGeneModelChars`	1	1
	`GenesBySnps`	1	1
	`GenesBySpanLogic`	1	1
	`DynSpansByMotifSearch`	1	1
Comparative genomics	`GenesByOrthologs`	15	1
	`GenesByOrthologPattern`	8	1
	`GenesByTaxon`	5	1
Phenotype & immunology	`GenesByPhenotype*`	1	1
	`GenesByAntibodyArray*`	1	1
	`GenesByReactionCompounds`	1	1

BIBLIOGRAPHY

Mohammad Aliannejadi, Hamed Zamani, Fabio Crestani, and W. Bruce Croft. 2019. Asking clarifying questions in open-domain information-seeking conversations. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 475–484.

Beatrice Amos, Cristina Aurrecoechea, Matthieu Barba, Ana Barreto, Evelina Y. Basenko, Wojciech Bażant, Robert Belnap, Ann S. Blevins, Ulrike Böhme, John Brestelli, Brian P. Brunk, Mark Caddick, Danielle Callan, Lahcen Campbell, Mikkel B. Christensen, George K. Christophides, Kathryn Crouch, Kristina Davis, Jeremy DeBarry, and 47 others. 2022. VEuPathDB: the eukaryotic pathogen, vector and host bioinformatics resource center. *Nucleic Acids Research*, 50(D1):D898–D911. PMC10767879.

Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. 2019. Giving BERT a Calculator: Finding Operations and Arguments with Reading Comprehension. *ArXiv*, abs/1909.00109.

Chinmaya Andukuri, Jan-Philipp Fränken, Tobias Gerstenberg, and Noah D Goodman. 2024. STaR-GATE: Teaching language models to ask clarifying questions. *arXiv preprint arXiv:2403.19154*.

Cristina Aurrecoechea, Ana Barreto, Evelina Y. Basenko, Jodi Brestelli, Brian P. Brunk, Steve Cade, Keith Crouch, Ryan Doherty, Dave Falke, Steve Fischer, Bindu Gajria, Omar S. Harb, Mark Heiges, Christiane Hertz-Fowler, Sufen Hu, John Iodice, Jessica C. Kissinger, Cris Lawrence, Wei Li, and 10 others. 2017. EuPathDB: the eukaryotic pathogen genomics database resource. *Nucleic Acids Research*, 45(D1):D581–D591. PMC6157018.

Cristina Aurrecoechea, Jodi Brestelli, Brian P. Brunk, Steve Fischer, Bindu Gajria, Xin Gao, Arnab Gingle, Greg Grant, Omar S. Harb, Mark Heiges, Fernan Innamorato, John Iodice, Jessica C. Kissinger, Eric T. Kraemer, Wei Li, John A. Miller, Vishal Nayak, Cary Pennington, Deborah F. Pinney, and 6 others. 2010. EuPathDB: a portal to eukaryotic pathogen databases. *Nucleic Acids Research*, 38(suppl_1):D415–D419. PMC2808945.

Evelina Y. Basenko, Jane A. Pulman, Achchuthan Shanmugasundram, Omar S. Harb, Keith Crouch, David Starns, Susanne Warrenfeltz, Cristina Aurrecoechea, Christian J. Stoeckert, Jessica C. Kissinger, David S. Roos, and Christiane Hertz-Fowler. 2018. FungiDB: An integrated bioinformatic resource for fungi and oomycetes. *Journal of Fungi*, 4(1):39. PMC5872342.

Manlian Bi, Zhijie Bao, Dongna Xie, Xiaohan Xie, Changxiao Yang, Tao Wang, Yongtian Wang, and Jiajie Peng. 2025. BioRAGent: natural language biomedical querying with retrieval-augmented multiagent systems. *Briefings in Bioinformatics*, 26(5):bbaf539.

Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. 2023. ChemCrow: Augmenting large-language models with chemistry tools.

Akshat Chaudhari, Janghoon Ock, and Amir Barati Farimani. 2025. Modular large language model agents for Multi-Task computational materials science. ChemRxiv preprint.

Jacob Cohen. 1968. Weighted kappa: Nominal scale agreement with provision for scaled disagreement or partial credit. *Psychological Bulletin*, 70(4):213–220.

Defense Advanced Research Projects Agency (DARPA). 2024. Broad agency announcement: Scientific Feasibility (SciFy). Technical Report HR001124S0013, Defense Advanced Research Projects Agency (DARPA), Information Innovation Office (I2O). Broad Agency Announcement.

Bowen Deng, Peichen Zhong, KyuJung Jun, Janosh Riebesell, Kevin Han, Christopher J. Bartel, and Gerbrand Ceder. 2023. CHGNet as a pretrained universal neural network potential for charge-informed atomistic modelling. *Nature Machine Intelligence*, page 1–11.

Sajib Acharjee Dip, Dipanwita Mallick, Uddip Acharjee Shuvo, Shovito Barua Soumma, Fazle Rafsani, Bikash Kumar Paul, Nazifa Ahmed Moumi, Shafayat Ahmed, and Liqing Zhang. 2026. Large language model agents for biological intelligence across genomics, proteomics, spatial biology, and biomedicine. *Briefings in Bioinformatics*, 27(2):bbag110.

Steve Fischer, Cristina Aurrecoechea, Brian P. Brunk, Xin Gao, Omar S. Harb, Eileen T. Kraemer, Cary Pennington, Charles Treatman, Jessica C. Kissinger, David S. Roos, and Christian J.

Stoeckert. 2011. The strategies WDK: a graphical search interface and web development kit for functional genomics databases. *Database*, 2011:bar027.

Gloria I. Giraldo-Calderón, Scott J. Emrich, Robert M. MacCallum, Gareth Maslen, Emmanuel Dialynas, Panagiotis Topalis, Nicholas Ho, Sandra Gesing, the VectorBase Consortium, Gregory Madey, Frank H. Collins, and Daniel Lawson. 2015. VectorBase: an updated bioinformatics resource for invertebrate vectors and other organisms related with human diseases. *Nucleic Acids Research*, 43(D1):D707–D713. PMC9133010 (2022 update).

Gilbert Henry Gulliver. 1913. *Metallic Alloys: Their Structure and Constitution*, 2 edition. Charles Griffin and Company, London. First edition 1908; Non-equilibrium solidification model presented in Chapter IX.

H. A. Jahn and E. Teller. 1937. Stability of polyatomic molecules in degenerate electronic states. i. orbital degeneracy. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 161(905):220–235.

Anubhav Jain, Shyue Ping Ong, Geoffroy Hautier, Wei Chen, William Davidson Richards, Stephen Dacek, Shreyas Cholia, Dan Gunter, David Skinner, Gerbrand Ceder, and Kristin A. Persson. 2013. The Materials Project: A materials genome approach to accelerating materials innovation. *APL Materials*, 1(1):011002.

Shuyi Jia, Chao Zhang, and Victor Fung. 2024. LLMatDesign: Autonomous materials discovery with large language models. *Preprint*, arXiv:2406.13163.

Aoyan Liang, Nicolas Bertin, Xinran Zhou, Sylvie Aubry, and Vasily Bulatov. 2025. Atomistic insights into solid solution strengthening: size misfit versus stiffness misfit.

Siyu Liu, Jiamin Xu, Beilin Ye, Bo Hu, David J. Srolovitz, and Tongqi Wen. 2025a. MatTools: Benchmarking large language models for materials science tools. *Preprint*, arXiv:2505.10852.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2024. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations (ICLR)*.

Yang Liu, Rongbo Shen, Lu Zhou, Qingyu Xiao, Jiao Yuan, and Yixue Li. 2025b. A data-intelligence-intensive bioinformatics copilot system for large-scale omics research and scientific insights. *Briefings in Bioinformatics*, 26(4):bbaf312.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Nikita Mehandru, Amanda K. Hall, Olesya Melnichenko, Yulia Dubinina, Daniel Tsirulnikov, David Bamman, Ahmed Alaa, Scott Saponas, and Venkat Sai Malladi. 2025. BioAgents: Bridging the gap in bioinformatics analysis with multi-agent systems. *Scientific Reports*, 15.

Richard Otis and Zi-Kui Liu. 2017. pycalphad: CALPHAD-based computational thermodynamics in Python. *Journal of Open Research Software*, 5(1):1.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. TALM: Tool Augmented Language Models. *ArXiv*, abs/2205.12255.

Erwin Povoden-Karadeniz. 2025. MatCalc thermodynamic database for aluminium alloys (`mc_al_-v2.037.tdb`). Open thermodynamic database file for MatCalc 6.00 or higher. Thermodynamic data for Al-based alloys; originally released 2023 and updated through 2025 (version 2.037); adapted for use with pycalphad in this work.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*.

Jose Luis Saenz-Garcia, Cristiane Lourenco Borges, Martha Noleto Boia, Nivedita Mudalige, and Jeroen P. Saeij. 2016. Identification of *Toxoplasma gondii* GAMA homologue. *Parasitology International*, 65(5 Pt A):463–470. PMC4863602.

N. Saunders and A. P. Miodownik. 1998. *CALPHAD (Calculation of Phase Diagrams): A Comprehensive Guide*. Elsevier / Pergamon.

Erich Scheil. 1942. Bemerkungen zur schichtkristallbildung. *International Journal of Materials Research*, 34:70 – 72.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551. Curran Associates, Inc.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2024. Quantifying language models' sensitivity to spurious features in prompt design. In *International Conference on Learning Representations (ICLR)*.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Jimin Sun, So Yeon Min, Yingshan Chang, and Yonatan Bisk. 2024. Tools Fail: Detecting silent errors in faulty tools. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14272–14289, Miami, Florida, USA. Association for Computational Linguistics.

L. Vegard. 1921. Die konstitution der mischkristalle und die raumfüllung der atome. *Zeitschrift für Physik*, 5:17–26.

VEuPathDB. Search strategies exercise (PlasmoDB). Hinxton Parasite Workshop.

VEuPathDB. 2016. Enrichment & transform exercise. Athens 2-Day Workshop, University of Georgia.

VEuPathDB. 2020. Strategies in PlasmoDB. VEuPathDB Training Module.

VEuPathDB. 2022. GO enrichment workshop exercise. VEuPathDB Workshop.

VEuPathDB. 2023. FungiDB GO enrichment exercise. Hinxton Fungal Workshop.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv: Arxiv-2305.16291*.

Yihang Xiao, Jinyi Liu, Yan Zheng, Xiaohan Xie, Jianye Hao, Mingzhi Li, Ruitao Wang, Fei Ni, Yuxiao Li, Jintian Luo, Shaoqing Jiao, and Jiajie Peng. 2024. CellAgent: An LLM-driven multi-agent framework for automated single-cell data analysis.

Tiantian Yang, Yihang Xiao, Zhijie Bao, Jianye Hao, and Jiajie Peng. 2025. The rise and potential opportunities of large language model agents in bioinformatics and biomedicine. *Briefings in Bioinformatics*, 26(6):bbaf601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Botao Yu, Frazier N. Baker, Ziru Chen, Garrett Herb, Boyu Gou, Daniel Adu-Ampratwum, Xia Ning, and Huan Sun. 2025. Tooling or not tooling? The impact of tools on language agents for chemistry problem solving. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 7620–7640, Albuquerque, New Mexico. Association for Computational Linguistics.

Huan Zhang, Yu Song, Ziyu Hou, Santiago Miret, and Bang Liu. 2024. HoneyComb: A flexible LLM-based agent system for materials science. In *Findings of the Association for Computational*

Linguistics: EMNLP 2024, pages 3369–3382, Miami, Florida, USA. Association for Computational Linguistics.

Kaizhong Zhang and Dennis Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262.

Andrew Zhu, Liam Dugan, Alyssa Hwang, and Chris Callison-Burch. 2023. Kani: A lightweight and highly hackable framework for building language model applications. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pages 65–77, Singapore. Association for Computational Linguistics.