RHIM: BENCHMARKING REDUNDANT HYPOTHESIS IDENTIFICATION REVEALS SYSTEMATIC GAPS IN LLM LOGICAL REASONING

Hai Dinh * † **Tuan Luong** * **Kha Pham** *

ABSTRACT

Identifying and removing redundant hypotheses is fundamental to mathematical discovery, yet the ability of large language models to perform this reasoning remains unexplored. We introduce RHIM (Redundant Hypothesis Identification in Mathematics), which is, to the best of our knowledge, the first benchmark for evaluating redundant hypothesis detection in mathematical proof problems, comprising 200 problems with verified ground truth. Through comprehensive experiments with state-of-the-art models — including DeepSeek-Reasoner, Gemini-2.5-Flash, and GPT-5.2 — we reveal critical failures across three hierarchical tasks: detection (false alarm rates 38–99.5%), identification (accuracy 32–64%, barely above the 25.8% random baseline imposed by variable hypothesis counts), and verification (23–34.5% acceptance of logically invalid proofs). These results demonstrate that proof generation ability does not imply understanding of logical dependencies between assumptions and conclusions — a capability essential for rigorous mathematical reasoning and theorem refinement.

1 INTRODUCTION

A central activity in mathematical research is not merely proving theorems, but refining them — stripping away unnecessary assumptions to reveal the minimal conditions under which a result holds. This process of *hypothesis minimization* is fundamental: a leaner theorem has broader applicability, exposes deeper logical structure, and is more amenable to generalization. Yet whether large language models (LLMs) can perform this kind of reasoning — determining which assumptions are *actually necessary* — remains, to the best of our knowledge, entirely unexplored.

Large language models have demonstrated remarkable capabilities in mathematical reasoning (Trinh et al., 2024; Guo et al., 2025), showing promise for both proving established theorems and assisting in mathematical discovery (Davies et al., 2021; Novikov et al., 2025; Jang & Ryu, 2025). However, existing benchmarks evaluate *forward* reasoning: given a set of assumptions $\mathcal{A}$ and a goal G, produce a valid proof that G follows from $\mathcal{A}$. This paper instead evaluates *backward* reasoning: given a proof problem $(\mathcal{A}, G)$, determine which elements of $\mathcal{A}$ are logically necessary. The distinction is non-trivial. A model can successfully prove G from $\mathcal{A}$ without ever checking whether $\mathcal{A} \setminus \{A_i\}$ also suffices — and our results show that current models, by default, do not perform this check.

To investigate this gap, we introduce RHIM (Redundant Hypothesis Identification in Mathematics), a benchmark of 200 mathematical proof problems with verified ground truth, which is, to the best of our knowledge, the first dataset constructed specifically for evaluating redundancy detection in mathematical proofs. Problems are drawn from Math Stack Exchange and constructed so that each contains exactly one injected redundant hypothesis, guaranteed unnecessary via the original community-accepted proof. We evaluate models on three hierarchical tasks: *detection* (does redundancy exist?), *identification* (which hypothesis is redundant?), and *verification* (can the model construct a valid proof without the redundant hypothesis?). The hierarchical design is deliberate — a model that cannot reliably detect redundancy should not be expected to identify or verify it, and failures at each level carry distinct diagnostic implications.

*Department of Mathematics and Informatics, Ho Chi Minh City University of Education.
†Correspondence to: Hai Dinh <4501101017@student.hcmue.edu.vn>

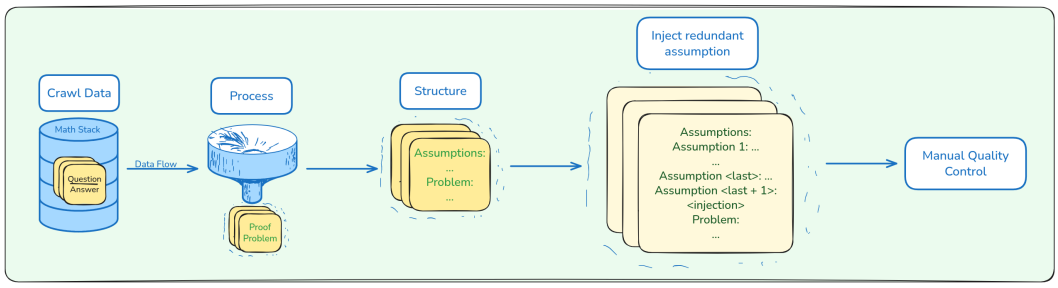

Figure 1: Overview of the five-stage dataset construction pipeline for RHIM: (1) crawling raw proof problems from Math Stack Exchange, (2) applying two successive filtering stages to yield well-defined proof problems with accepted answers and unambiguous goal statements, (3) reformulating each problem into a canonical structured form that explicitly separates enumerated assumptions from the goal statement, (4) injecting a redundant assumption by extracting an intermediate statement from the accepted proof and appending it in ordinal order to the assumption set—guaranteeing that every injected hypothesis is provably unnecessary via the original proof, and (5) manual quality control verifying display-math formulation, domain accessibility, structural fidelity, and logical consistency, producing high-confidence problems with verified redundancy annotations suitable for rigorous LLM evaluation. Representative examples from RHIM are provided in Appendix A.1.

Our experiments reveal a striking and consistent pathology across all evaluated models: a systematic *yes-bias* in redundancy detection. All models achieve perfect recall (1.000) — they almost never conclude that a problem is redundancy-free — but do so at the cost of false alarm rates between 38% and 99.5% on non-redundant problems. This suggests models treat the question "does a redundancy exist?" as effectively rhetorical, defaulting to an affirmative response rather than performing the logical sufficiency analysis the task requires. Critically, this bias persists even when models are embedded in agentic pipelines with explicit task decomposition and proof verification stages, indicating that it is embedded in the underlying models rather than an artifact of prompting.

The failures compound across tasks. When redundancy actually exists, identification accuracy reaches only 32–64% — barely above the 25.8% random baseline, which itself reflects the variable number of hypotheses per problem (1–8) rather than a binary choice. Most critically, even state-of-the-art reasoning models prove unreliable as automated proof verifiers, accepting 23–34.5% of logically invalid proofs on control problems where redundancy is provably impossible by construction. This unreliability directly undermines the use of LLMs as proof checkers in automated mathematical workflows.

These results expose a fundamental asymmetry: models achieve strong performance on standard forward-reasoning benchmarks while failing dramatically at the backward reasoning required for redundancy analysis. This asymmetry suggests that current LLMs rely on pattern matching over proof structure rather than genuine logical reasoning about the sufficiency relationships between assumptions and goals. Theorem refinement — a core activity of mathematical research — thus remains beyond the reliable capability of present systems.

Our main contributions are twofold. First, we introduce RHIM, which is, to the best of our knowledge, the first benchmark for redundant hypothesis identification in mathematical proofs, comprising 200 problems with verified ground truth spanning algebra, number theory, analysis, combinatorics, and topology. Second, through comprehensive experiments across reasoning, non-reasoning, and agentic LLM configurations, we characterize a systematic yes-bias that is universal across architectures — manifesting as perfect recall but 38–99.5% false alarm rates, near-chance identification accuracy (32–64% against a 25.8% random baseline), and 23–34.5% acceptance of logically invalid proofs — demonstrating that neither solver-verifier pipelines nor task-decomposition architectures resolve the underlying deficiency. Together, these contributions establish RHIM as a concrete testbed for measuring progress toward genuine logical reasoning in language models.

2 Related Work

Early mathematical benchmarks like MATH and GSM8k (Hendrycks et al., 2021; Cobbe et al., 2021) evaluated models through final answer accuracy—a pragmatic choice since reliably verifying correctness and necessity of individual proof steps remains challenging. Yet this approach's limitations have become evident: models increasingly produce correct answers despite flawed intermediate reasoning (Shao et al., 2025; Turpin et al., 2023). Contemporary work has therefore shifted toward process-oriented evaluation, using step-level rewards (Lightman et al., 2023) in reinforcement learning (Guo et al., 2025) to encourage sound reasoning over pattern-matching. However, the fundamental challenge persists: mathematical proofs admit infinitely many valid formulations with diverse logical structures reaching identical conclusions, making verification of reasoning quality—not just correctness—inherently difficult.

Within this landscape, a critical gap remains: systematic evaluation of whether models understand which assumptions are necessary. Hypothesis minimization—identifying redundant assumptions to reveal essential logical structure—is fundamental to mathematical practice (Stillwell, 2018), yet existing work emphasizes forward inference over backward analysis of premise necessity (Ren et al., 2025). While some benchmarks evaluate logical reasoning through natural language inference or puzzles (Parmar et al., 2024; Zheng et al., 2021), none systematically assess identification of superfluous assumptions in formal mathematical contexts. This gap is consequential: evidence suggests LLMs struggle with counterfactual reasoning and distinguishing correlation from causation (Kıcıman et al., 2024), indicating potential weaknesses in reasoning about hypothesis necessity.

RHIM addresses this gap through a novel evaluation paradigm: testing whether models distinguish necessary from superfluous premises. Unlike benchmarks that evaluate forward reasoning—producing valid proofs from given assumptions—we evaluate backward reasoning: determining which assumptions are actually required. This requires meta-level reasoning about problem structure rather than merely applying proof techniques. Our experiments reveal a striking asymmetry: models achieve strong performance on standard forward-reasoning tasks while failing dramatically at redundancy detection. This asymmetry exposes fundamental limitations in current LLMs' logical reasoning capabilities, demonstrating that proof generation ability does not imply understanding of logical dependencies between assumptions and conclusions.

3 Preliminaries

We establish the formal framework for analyzing redundant hypotheses in mathematical proof problems. This formalization provides the theoretical foundation for our benchmark construction and evaluation methodology.

3.1 Mathematical Proof Problems and Hypothesis Redundancy

Definition 3.1 (Proof Problem). *A mathematical proof problem $\mathcal{P}$ is a tuple $(\mathcal{A}, G)$ where $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ is a finite set of assumptions (hypotheses), each representing a mathematical statement or condition, and G is the goal statement to be proven. A solution to $\mathcal{P}$ is a valid mathematical proof demonstrating that G follows from $\mathcal{A}$ through sound logical inference.*

The notion of redundancy captures whether an assumption is essential for establishing the goal statement.

Definition 3.2 (Redundant Hypothesis). *Let $\mathcal{P} = (\mathcal{A}, G)$ be a proof problem with $\mathcal{A} = \{A_1, \ldots, A_n\}$. An assumption $A_i \in \mathcal{A}$ is* redundant *if and only if there exists a valid proof of G using only the reduced assumption set $\mathcal{A} \setminus \{A_i\}$.*

Equivalently, A_i is redundant when its removal does not affect the provability of G. We denote the set of all redundant hypotheses in $\mathcal{P}$ as $\mathcal{R}(\mathcal{P}) \subseteq \mathcal{A}$.

Remark 3.1 (Redundancy Types and Problem Scope). Definition 3.2 encompasses two types: assumptions derivable from $\mathcal{A} \setminus \{A_i\}$, and assumptions unnecessary for proving G even if not derivable. For unambiguous evaluation, we restrict to problems with exactly one redundant assumption ($|\mathcal{R}(\mathcal{P})| = 1$), identified as intermediate results in community-validated proofs.

3.2 Evaluation Tasks

We formalize three hierarchical tasks for assessing LLM capabilities in redundancy reasoning:

Task 3.1 (Detection)**.** Given $\mathcal{P} = (\mathcal{A}, G)$, determine whether redundancy exists, $|\mathcal{R}(\mathcal{P})| > 0$.

Task 3.2 (Identification)**.** Given $\mathcal{P} = (\mathcal{A}, G)$ where $|\mathcal{R}(\mathcal{P})| > 0$, identify $A_i \in \mathcal{R}(\mathcal{P})$.

Task 3.3 (Verification)**.** Given $\mathcal{P} = (\mathcal{A}, G)$ and a hypothesis $A_i \in \mathcal{A}$ claimed to be redundant, construct a valid proof of G from $\mathcal{A} \setminus \{A_i\}$, thereby verifying $A_i \in \mathcal{R}(\mathcal{P})$ per Definition 3.2.

Task 3.3 addresses the limitation that our ground truth captures only one redundant hypothesis per problem, while models may identify alternative valid redundancies. By requiring formal proofs, we can validate such identifications, which we evaluate in Experiment 5.3.

4 The RHIM Benchmark

We introduce RHIM (Redundant Hypothesis Identification in Mathematics), a benchmark comprising 200 with mathematical proof problems exactly one injected redundant hypothesis—designed to evaluate whether large language models can detect and reason about hypothesis redundancy. Each redundant hypothesis is verifiably unnecessary through community-validated proofs, ensuring ground truth validity. The benchmark is constructed through a systematic five-stage pipeline balancing validity guarantees with mathematical diversity.

4.1 Dataset Construction Pipeline

Our construction methodology ensures both validity of redundant hypotheses and diversity of mathematical content. Figure 1 illustrates the complete pipeline.

Stage 1: Data Collection and Preprocessing. We retrieve 11,776 mathematical problems from Math Stack Exchange using its official API. To ensure quality and relevance, we retain only problems tagged as proof-based questions, require the presence of an accepted answer (community-validated solution), and exclude problems containing figures or diagrams that cannot be processed programmatically. This filtering yields an initial corpus of high-quality proof problems suitable for further processing.

Stage 2: Proof Problem Filtering. Not all problems passing Stage 1 are appropriate for redundancy analysis. We apply content-based filtering to exclude problems requesting counterexamples or verification tasks, open-ended exploration problems without clear proof objectives, and problems with ambiguous or underspecified goal statements. After this stage, 3,439 well-defined proof problems remain, each with a clear set of assumptions and a goal statement.

Stage 3: Problem Structuring. Mathematical problems on Math Stack Exchange are presented in natural language with varying formats, making systematic hypothesis manipulation infeasible without normalization. We therefore reformulate each problem into a canonical structured form (see Appendix A.2) using DeepSeek-R1, explicitly separating enumerated assumptions from the goal statement. This structuring is crucial for subsequent redundancy injection and ensures consistent problem representation across the benchmark. The complete prompt is provided in Appendix A.3.

Stage 4: Redundant Assumption Injection. This stage constructs problems satisfying $|\mathcal{R}(\mathcal{P})| = 1$ by leveraging accepted solutions. The procedure operates as follows. First, from each accepted answer, we extract intermediate mathematical statements using regex-based rules (detailed in Appendix A.5). These rules prioritize expressions enclosed in mathematical delimiters (`$$...$$`) and statements containing logical or relational symbols ($=$, $\neq$, $<$, $\leq$, $\in$, $\subseteq$, etc.). Second, an extracted statement S becomes a redundant hypothesis candidate if it appears as an intermediate result in the proof. By construction, if S is proven using the original assumptions $\mathcal{A}$, then adding S to $\mathcal{A}$ creates a redundant hypothesis—the proof itself witnesses that the goal G is provable from $\mathcal{A}$ alone. Third, we select exactly one candidate statement per problem and inject it into the structured assumption set, creating $\mathcal{A}' = \mathcal{A} \cup \{S\}$. The augmented problem $\mathcal{P}' = (\mathcal{A}', G)$ now contains a known redundant hypothesis S.

Table 1: Redundancy detection and identification performance across reasoning, non-reasoning, and agentic pipeline configurations. Detection metrics (Det. Acc, Det. Prec, Det. Rec, Det. F1, Spec, FAR) assess binary classification of whether a problem contains a redundant hypothesis, while ExM and UIA measure exact match and unconditional identification accuracy respectively. All models achieve perfect recall (1.000) but vary widely in specificity (0.005–0.730) and identification accuracy (0.321–0.626), revealing a universal yes-bias where detection strength does not guarantee correct hypothesis localization. ↑ indicates higher is better, ↓ indicates lower is better.

Mode	Model	Det. Acc ↑	Det. Prec ↑	Det. Rec ↑	Det. F1 ↑	FAR ↓	Spec ↑	ExM ↑	UIA ↑
Reasoning	DSR	0.775	0.690	1.000	0.816	0.450	0.550	0.668	0.541
	GPT-5.2	0.814	0.728	1.000	0.843	0.370	0.630	0.724	0.596
Non-reasoning	Qwen3-80B	0.675	0.606	1.000	0.755	0.650	0.350	0.588	0.500
	DeepSeek-Chat	0.502	0.501	1.000	0.668	0.995	0.005	0.322	0.321
	Gemini-2.5-Flash	0.795	0.709	1.000	0.830	0.410	0.590	0.718	0.599
Agentic P.1	DS-Chat + DSR	0.804	0.717	1.000	0.835	0.390	0.610	0.696	0.562
	Gemini-2.5-F + DSR	0.865	0.787	1.000	0.881	0.270	0.730	0.681	0.498
Agentic P.2	DS-Chat + DSR	0.838	0.755	1.000	0.860	0.325	0.675	0.752	0.626
	Gemini-2.5-F + DSR	0.785	0.699	1.000	0.823	0.430	0.570	0.708	0.591

Abbreviations—Qwen3-80B: Qwen3-Next-80B-A3B-Instruct; DSR: DeepSeek-Reasoner; DS-Chat: DeepSeek-Chat; Gemini-2.5-F: Gemini-2.5-Flash; Agentic P.1: Agentic Pipeline 1; Agentic P.2: Agentic Pipeline 2.

This construction ensures ground truth validity: every redundant hypothesis in RHIM is verifiably redundant via the original accepted proof.

Stage 5: Manual Quality Control. We apply human verification to ensure benchmark quality across four criteria. First, we retain only problems where injected hypotheses appear in display math (`$$...$$`) rather than inline math (`$...$`), as display-mode statements represent substantial mathematical results rather than notational fragments. Second, we filter problems requiring specialized mathematical background beyond general graduate-level training—problems incomprehensible even after careful reading of question and accepted answer indicate domain specificity unsuitable for general LLM evaluation. Third, we verify that structured reformulation preserves the original problem's assumption count, removing cases where LLM structuring introduced spurious assumptions or omitted essential ones. Fourth, we validate logical consistency between original and structured forms, ensuring reformulation does not alter mathematical content. This manual curation produces 200 high-confidence problems with verified redundancy annotations suitable for rigorous evaluation.

4.2 Dataset Characteristics

Scale and Diversity. RHIM comprises 200 manually verified problems spanning diverse mathematical domains (see Figure 2 and Table 5). The benchmark emphasizes core topics including real analysis (21.5%), sequences and series (14.5%), calculus (13.5%), inequalities (12.5%), and linear algebra (11.0%), with substantial coverage of integration, abstract algebra, measure theory, complex analysis, number theory, and topology. These problems represent careful curation from 3,439 structured candidates, with manual filtering selecting 200 problems (32.4% acceptance rate). This conservative selection prioritizes quality over quantity, establishing a high-confidence evaluation set while suggesting future expansion to approximately 1,100 problems from the remaining candidate pool.

Quality Assurance. Each problem satisfies four properties: well-formedness (clear separation between assumptions and goal), provability (existence of community-accepted proof), ground truth validity (constructive proof that the redundant hypothesis is indeed redundant), and single redundancy (exactly one redundant hypothesis per problem).

Data Schema. The benchmark contains two primary fields: `Original_Problem_with_numerical_assumption` stores the original structured problem with numerically ordered assumptions, while `Problem_with_redundant_assumption` contains the augmented problem with one injected redundant hypothesis, also with numerically ordered assumptions. Complete schema details including metadata fields are provided in Appendix A.6.

Table 2: Exact match accuracy stratified by the number of hypotheses in each problem, reported across reasoning, non-reasoning, and agentic pipeline configurations. Each column corresponds to a hypothesis-count bucket with its sample size in parentheses, reflecting the natural distribution of the benchmark (bulk at 2–3 hypotheses, $n = 220$). Performance does not degrade monotonically with hypothesis count; instead, a consistent trough appears at 4 hypotheses across nearly all models and pipelines, indicating a structural difficulty threshold rather than a simple scaling effect. Results for 6–8 hypotheses ($n \leq 14$) carry wide confidence intervals and should not be over-interpreted.

Mode	Model	1 (n=24)	2 (n=92)	3 (n=128)	4 (n=92)	5 (n=42)	6 (n=14)	7 (n=6)	8 (n=2)
Reasoning	DeepSeek-Reasoner	0.708	0.728	0.625	0.641	0.690	0.714	0.667	0.500
	GPT-5.2	0.828	0.808	0.691	0.613	0.714	0.786	1.000	0.000
Non-reasoning	DeepSeek-Chat	0.042	0.152	0.352	0.467	0.357	0.500	0.333	1.000
	Qwen3-80B	0.333	0.522	0.641	0.598	0.667	0.643	0.667	0.500
	Gemini-2.5-Flash	0.583	0.739	0.734	0.674	0.714	0.929	0.833	0.500
Agentic P.1	DS-Chat + DSR	0.828	0.721	0.675	0.570	0.762	0.867	1.000	0.000
	Gemini-2.5-F + DSR	0.759	0.796	0.626	0.526	0.775	0.733	0.800	0.000
Agentic P.2	DS-Chat + DSR	0.828	0.712	0.782	0.712	0.667	1.000	1.000	1.000
	Gemini-2.5-F + DSR	0.793	0.663	0.742	0.637	0.714	0.867	1.000	0.000

4.3 Evaluation Protocol

Evaluation Data. We evaluate on 200 manually verified problems from RHIM and 200 original problems without redundancy. This balanced design prevents trivial solutions and enables measurement of both detection precision (avoiding false alarms on non-redundant problems) and recall (catching actual redundancy). Random baseline performance is 25.8% due to variable hypothesis counts (1–8 per problem).

Metrics. We employ task-specific metrics corresponding to Tasks 3.1–3.3. Detection (Task 3.1) uses accuracy, precision, recall, F1-score, specificity (**Spec**↑), and false alarm rate (**FAR**↓). Identification (Task 3.2) measures unconditional identification accuracy (**UIA**↑)—correct hypothesis identification across all 400 problems, penalizing false positives on non-redundant cases—alongside exact match accuracy (**ExM**↑). Verification (Task 3.3) assesses proof validity through automated checking using DeepSeek-Reasoner as verifier, evaluated on control problems where redundancy is provably impossible.

Two-Phase Evaluation Strategy. We first test models on problems *without* redundancy to establish baseline reliability and measure false positive rates—if models incorrectly identify redundancies in original problems, this indicates systematic bias. Only after characterizing this bias do we evaluate performance on problems with injected redundant hypotheses. This protocol reveals whether models genuinely understand redundancy or merely pattern-match surface cues.

5 Experiments and Results

5.1 Experimental Setup

Models. We evaluate the following state-of-the-art LLMs on mathematical reasoning benchmarks AIME. The reasoning models include (DeepSeek-V3 .2 Thinking Mode) and GPT-5.2 (reason medium), while the non-reasoning models comprise DeepSeek-Chat (DeepSeek-V3.2 Non-thinking Mode), Gemini-2.5-Flash, and Qwen3-Next-80B-A3B-Instruct.

Evaluation Data. All experiments are conducted on the manually verified subset of 200 problems from RHIM, which provides high-confidence ground truth for redundancy annotations. To assess model reliability, we employ a two-phase evaluation strategy: we first test models on these problems *without* redundant assumptions to measure false positive rates—if models incorrectly identify redundancies in original problems, this indicates unreliable reasoning or bias toward user expectations. Only after establishing baseline reliability do we evaluate performance on problems with injected redundant assumptions. Random baseline performance is 25.8% due to variable hypothesis counts.

Table 3: Error distribution breakdown across reasoning, non-reasoning, and agentic pipeline configurations, categorized into three mutually exclusive error types: False Positive (FP, claimed redundancy when none exists), False Negative (FN, missed actual redundancy), and Wrong Hypothesis (WH, detected redundancy but localized the wrong hypothesis). Percentages within FP, FN, and WH columns indicate each type's share of total errors for that model. Strikingly, false negatives are entirely absent across all models (FN=0), while false positives dominate (42.4–78.8%), confirming a universal yes-bias; the remaining WH errors reveal that even correct detection frequently fails at precise localization, with Agentic P.1 Gemini-2.5-F+DSR exhibiting the most severe localization failure (WH=57.6%).

Mode	Model	Total Errors	FP	FN	WH	Error Rate
Reasoning	DeepSeek-Reasoner	133	90 (67.7%)	0 (0.0%)	43 (32.3%)	33.2%
	GPT-5.2	110	74 (67.3%)	0 (0.0%)	36 (32.7%)	27.6%
Non-reasoning	Qwen3-80B	165	130 (78.8%)	0 (0.0%)	35 (21.2%)	41.2%
	DeepSeek-Chat	271	199 (73.4%)	0 (0.0%)	72 (26.6%)	67.8%
	Gemini-2.5-Flash	113	82 (72.6%)	0 (0.0%)	31 (27.4%)	28.2%
Agentic P.1	DS-Chat + DSR	121	78 (64.5%)	0 (0.0%)	43 (35.5%)	30.4%
	Gemini-2.5-F + DSR	125	53 (42.4%)	0 (0.0%)	72 (57.6%)	31.9%
Agentic P.2	DS-Chat + DSR	99	65 (65.7%)	0 (0.0%)	34 (34.3%)	24.8%
	Gemini-2.5-F + DSR	117	86 (73.5%)	0 (0.0%)	31 (26.5%)	29.2%

5.2 Experiment 1: Redundant Hypothesis Detection and Identification

Methodology. We evaluate models on Tasks 3.1 and 3.2 using a single structured prompt that requests both outputs sequentially. The prompt instructs the model to determine whether the problem contains a redundant hypothesis (yes/no), identify which specific assumption is redundant if applicable, and provide justification for the determination. The prompt is provided in Appendix A.4.

Results. Table 1 reveals two critical failures. First, all models exhibit systematic bias toward claiming redundancy: perfect recall (1.000) but poor specificity (0.005–0.620), with false alarm rates of 38–99.5%. DeepSeek-Chat demonstrates extreme bias (FAR=0.995), while even top performers GPT-5.2 and Gemini-2.5-Flash misclassify 38–41% of non-redundant problems. Second, unconditional identification accuracy collapses to 32.1–63.8%—barely above the 25.8% random baseline—with GPT-5.2 achieving the best performance (63.8%) and DeepSeek-Chat falling below chance (32.1%).

Analysis. The results demonstrate that models are strongly biased toward claiming redundancy (perfect recall, FAR=38–99.5%) yet cannot reliably identify which hypothesis is actually redundant (unconditional identification 32–64%). This two-level failure suggests models exploit superficial problem-structure cues to trigger "redundancy exists" responses without performing the rigorous logical sufficiency analysis required to pinpoint the redundant assumption. A system that indiscriminately flags 38–99.5% of problems as redundant while achieving near-chance identification accuracy provides limited practical utility for redundancy detection tasks.

Stratified Analysis: Hypothesis Count Sensitivity. In the Table 2, performance across hypothesis counts shows irregular patterns rather than expected monotonic degradation, with Gemini-2.5-Flash peaking at 6 hypotheses (92.9%, n=14) and the extreme values at 8 hypotheses (50-100%, n=2) reflecting statistical noise from insufficient samples rather than genuine model capabilities, while the bulk of problems (220/400 with 2-3 hypotheses) show relatively stable performance (62.5-73.9%) across models.

Error Analysis: Wrong Hypothesis Identification. Show in Table 3, false positives dominate all error distributions (67.7–73.4% of total errors) while false negatives are entirely absent (FN=0), indicating models default to "when in doubt, claim redundancy" rather than performing balanced logical analysis.

Table 4: DeepSeek-Reasoner reviewer performance on proof verification across both agentic pipeline configurations, evaluated on a held-out control set of problems without redundancy. Metrics include FAR_{rv} (false acceptance rate of invalid proofs), TNR_{rv} (true negative rate), and standard classification metrics (Acc, Rec, Prec), where subscript $_{rv}$ denotes reviewer-level evaluation distinct from the detection metrics in Table 1. Pipeline 2 with DeepSeek-Chat achieves the lowest false acceptance rate (FAR_{rv}=0.230) and highest precision (0.745), while Pipeline 2 with Gemini-2.5-Flash degrades substantially (FAR_{rv}=0.345), suggesting that the task-decomposition architecture benefits reviewer reliability only when paired with a compatible solver. ↑ indicates higher is better, ↓ indicates lower is better.

Pipeline	Model	FAR_{rv} ↓	TNR_{rv} ↑	Acc_{rv} ↑	Rec_{rv} ↑	$Prec_{rv}$ ↑
Pipeline 1	DeepSeek-Chat + DeepSeek-Reasoner	0.300	0.700	0.650	0.610	0.670
(Solver-Verifier)	Gemini-2.5-Flash + DeepSeek-Reasoner	0.280	0.720	0.720	0.730	0.720
Pipeline 2	DeepSeek-Chat + DeepSeek-Reasoner	0.230	0.770	0.722	0.675	0.745
(Task-Decomposition)	Gemini-2.5-Flash + DeepSeek-Reasoner	0.345	0.655	0.612	0.570	0.622

5.3 Experiment 2: Agentic Pipeline with Proof Verification

Motivation. Direct evaluation of redundancy identification is limited because models may correctly identify alternative redundant hypotheses not captured by our ground truth. To address this, we implement agentic pipelines where models must prove their detected hypotheses are redundant (Task 3.3), then evaluate proof validity using DeepSeek-Reasoner as an automated verifier, calibrated on control problems without redundancy.

Pipeline Architectures. We implement two pipeline variants. Pipeline 1 (Solver-Verifier) uses a single solver agent followed by a separate verifier, while Pipeline 2 (Task-Decomposition) decomposes reasoning into specialized agents for detection, planning, and proof writing, followed by verification. Complete role specifications and prompts are in Appendix A.9.

Result: Detection and Identification. Table 1 shows that both pipelines inherit the yes-bias from Experiment 1, achieving perfect recall (1.000) at the cost of elevated false alarm rates (0.270–0.430) and poor specificity (0.570–0.730). Pipeline 2 with DS-Chat+DSR attains the highest exact match (0.752) and unconditional identification accuracy (0.626), while Pipeline 1 with Gemini-2.5-F+DSR leads on detection metrics (Det. Acc=0.865, F1=0.881, FAR=0.270) yet yields the lowest identification accuracy (UIA=0.498)—the widest detection–identification gap across all configurations.

Analysis. The inversion between detection and identification performance exposes a structural limitation of the solver-verifier architecture: a stronger solver produces more confident redundancy claims, which improves detection statistics while simultaneously misleading the downstream reviewer on hypothesis localization. Pipeline 2's task-decomposition design partially mitigates this by forcing explicit reasoning decomposition before identification, yielding more balanced performance across both levels. Nevertheless, the persistent yes-bias and sub-0.63 unconditional identification accuracy across all configurations confirm that architectural augmentation alone cannot overcome the shallow redundancy heuristics embedded in the underlying models.

Stratified Analysis: Hypothesis Count Sensitivity. As shown in Table 2, both pipelines exhibit irregular rather than monotonically degrading performance across hypothesis counts, with a consistent trough at 4 hypotheses (0.526–0.712) before partial recovery at 5, marking it as a structural difficulty threshold rather than a scaling artifact. Pipeline 2 maintains comparatively stable mid-range performance (0.637–0.782 for 2–4 hypotheses), whereas Pipeline 1 degrades more sharply, suggesting task decomposition better preserves reasoning coherence under growing hypothesis load. Extreme values beyond 6 hypotheses ($n \leq 14$) carry wide confidence intervals and should not be over-interpreted.

Error Analysis: Wrong Hypothesis Identification. As shown in Table 3, false positives dominate all error distributions (42.4–73.5%) while false negatives remain entirely absent, confirming that the yes-bias inherited from Experiment 1 persists structurally through both pipeline architectures. The starkest anomaly lies in Pipeline 1 with Gemini-2.5-F+DSR, where WH errors constitute 57.6% of total errors—the highest across all configurations—revealing that superior detection does

not imply accurate localization. Pipeline 2 achieves the lowest overall error rates (24.8–29.2%), indicating that task decomposition suppresses over-detection without resolving its root cause.

Results: Verification. Both pipelines inherit the systematic over-prediction bias from Experiment 5.1, achieving perfect recall but 27–43% false alarm rates (Table 1). Most critically, Table 4 shows that DeepSeek-Reasoner accepts 23–34.5% of logically invalid proofs on control problems where redundancy is impossible by construction, achieving only 61.2–72.2% verification accuracy despite being a state-of-the-art reasoning model.

Analysis. The results reveal two critical limitations. First, architectural changes cannot overcome the fundamental yes-bias: both pipelines achieve perfect recall but 27–43% false alarm rates, demonstrating that over-prediction is embedded in the models rather than prompt design. Second, state-of-the-art reasoning models cannot serve as reliable automated proof verifiers, accepting 23–34.5% of logically invalid proofs even where redundancy is provably impossible. This unreliability undermines model-generated proofs and necessitates human oversight or formal verification tools for high-stakes mathematical reasoning applications.

6 Discussion and Conclusion

We introduced RHIM, the first benchmark for evaluating redundant hypothesis detection in mathematical proofs, and demonstrated that state-of-the-art LLMs exhibit critical failures across detection, identification, and verification tasks. Models show systematic bias toward predicting redundancy exists (achieving perfect recall but 38–99.5% false alarm rates), cannot reliably identify which specific hypothesis is redundant (32–64% accuracy, barely above 25.8% random baseline), and prove fundamentally unreliable as proof verifiers (accepting 23–34.5% of logically invalid proofs). These failures reveal that despite strong performance on standard mathematical benchmarks, current LLMs lack the rigorous logical reasoning required to analyze sufficiency relationships between assumptions and goals.

Our findings have significant implications for deploying LLMs in mathematical research. The inability to reliably detect redundant hypotheses—a reasoning pattern central to theorem refinement and mathematical discovery—suggests that models rely on pattern matching rather than sound deductive reasoning. Future work should extend our methodology to multi-hypothesis redundancy and explore integration with formal verification tools. RHIM provides a concrete testbed for measuring progress toward genuine logical reasoning capabilities in language models.

References

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, et al. Advancing mathematics by guiding human intuition with ai. *Nature*, 600(7887):70–74, 2021.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Uijeong Jang and Ernest K Ryu. Point convergence of nesterov's accelerated gradient method: An ai-assisted proof. *arXiv preprint arXiv:2510.23513*, 2025.

Emre Kıcıman, Robert Ness, Amit Sharma, and Chenhao Tan. Causal reasoning and large language models: Opening a new frontier for causality, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. Logicbench: Towards systematic evaluation of logical reasoning ability of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13679–13707, 2024.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Zhihong Shao, Yuxiang Luo, Chengda Lu, ZZ Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. Deepseekmath-v2: Towards self-verifiable mathematical reasoning. *URL https://arxiv. org/abs/2511.22570*, 2025.

John Stillwell. Reverse mathematics: proofs from the inside out. 2018.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965, 2023.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

A APPENDIX

A.1 SAMPLES FROM RHIM

Linear Algebra Problem with redundant assumption (Assumption 3)

Assumption:
Assumption 1: n is a positive integer,
Assumption 2: n approaches infinity.
Assumption 3:

$$\sqrt[n]{n!} \geq \frac{n}{e} \to \infty.$$

Problem:
Prove that

$$\lim_{n \to +\infty} \sqrt[n]{n!}$$

is infinite.

Abstract Algebra Problem with redundant assumption (Assumption 5)

Assumption:
Assumption 1: G is an abelian group,
Assumption 2: $a \in G$ has order m,
Assumption 3: $b \in G$ has order n,
Assumption 4: m and n are positive integers.
Assumption 5:

$$\mathrm{X}, \mathrm{Y} \in \mathrm{G} \;\Rightarrow\; \exists\, \mathrm{Z} \in \mathrm{G} \colon\; \mathrm{o}(\mathrm{Z}) = \mathrm{lcm}(\mathrm{o}(\mathrm{X}), \mathrm{o}(\mathrm{Y}))$$

Problem:
Prove that there exists an element in G whose order is the least common multiple of m and n.

A.2 STRUCTURED PROBLEM

```
Assumption:
Assumption 1: <Assumption 1>,
Assumption 2: <Assumption 2>,
<...>
Assumption n: <Assumption n>.
Problem:
<Requirement of the problem>
```

Listing 1: Structure of a problem

A.3 PROMPT TO STRUCTURE THE PROBLEM

```
Act as a knowledgeable, and bilingual (Vietnamese and English)
    mathematics professor. Your task is to read the mathematics question
    on Math Stack Exchange, filtered the unrelated informations, and
    structure that problem as the following format. You will be given
    some example which I structured before.

To answer the question. You must follow below format:
```

```
###BEGIN_OF_FORMAT###
Assumption: <All the assumptions of the problem should be listed line by
    line. Each assumption addresses **only one property or condition**,
    applied to **one object**>
Problem: \<The problem\>
###

Example:
Original problem:
Prove that for every integer $n$, the number
$$A(n) = 5^n \left( 5^n + 1 \right) - 6^n \left( 3^n + 2^n \right)$$
is divisible by 91.

Formatted problem:
assumption:
$n$ is an integer,
$n$ is divisible by $7$.
problem:
Prove that $$A(n) = 5^n \left( 5^n + 1 \right) - 6^n \left( 3^n + 2^n \
    right)$$
is divisible by 91.
```

Listing 2: Prompt to structure the problems

A.4 PROMPT TO DETECT REDUNDANT ASSUMPTION

The prompt was designed to simulate the role of a knowledgeable and English mathematics professor, ensuring clarity and precision in the responses. The structure of the system prompt is provided below:

```
Act as a direct, efficient, knowledgeable, and English mathematics
professor. Your task is to read the structured mathematics problem and
answer the following question.

Question: Does this problem have redundant assumption?

To answer the question, you must follow the format below:
###BEGIN_OF_FORMAT###
Answer: <yes/no>

Ordinal number of redundant assumption:
<If the problem has a redundant assumption, write the ordinal number of
    redundant assumption, which is the same as the ordinal number of
    assumption in the problem. If the problem does not have a redundant
    assumption, write -1>

Redundant assumption:
<If the problem has a redundant assumption, write the redundant
    assumption here, write the same as assumption the problem
has. If the problem does not have a redundant assumption, write no>

Your explanation:
<if you suppose the problem has at least one redundant assumption, your
    explanation must include a solution (start the proof with **PROOF**)
    for the problem which don't use the redundant assumption. If the
    problem does not have a redundant assumption, your explanation must
    be The problem does not have a redundant assumption.>
###END_OF_FORMAT###
```

Listing 3: Detect redundant assumption

A.5 Assumption Extraction Rules

Regex-based Redundant Assumption Extraction

- **Primary pattern:** `$$...$$` (expression enclosed in four dollar signs)
- **Fallback pattern:** ... with logical/mathematical symbols such as $=, \neq, >, <, \geq, \leq, \in, \notin, \subset, \subseteq, \supset, \supseteq, |, \nmid, \forall, \exists, \rightarrow, \Rightarrow, \iff$
- If no valid expression is matched, the sample is discarded

A.6 Dataset Schema

Our benchmark is structured as a comprehensive Excel file with the following columns:

- `Link_API`: API endpoints for programmatic access to questions via Math Stack Exchange
- `Title`: Descriptive titles of the mathematical problems
- `Score`: Community-assigned relevance metrics (vote counts)
- `Category`: Mathematical domain classifications
- `Tags`: Topic-specific identifiers associated with each problem
- `Link`: Web browser access URLs to original questions
- `Content`: Textual content of questions (excluding visual elements)
- `AcceptedAnswer`: Community-validated solutions to each problem
- `llm_answer_create_structured_problem`: DeepSeekR1's formatted problem reconstructions
- `reasoning_create_structured_problem`: DeepSeekR1's logical processes for problem structuring
- `Original_Problem_with_numerical_assumption`: Formalized problem representations extracted from DeepSeekR1's outputs. The assumptions of problem were indexed numerically after.
- `Proof_problem`: Final classification markers for proof-based problems
- `Redundant_assumption`: Identified superfluous conditions (exactly one per problem)
- `Problem_with_redundant_assumption`: Augmented problem statements containing injected redundant assumptions. The assumptions of problem were indexed numerically after.

A.7 TAGS WORD CLOUD AND STATISTICS

The tags for 200 problems with redundancy from RHIM were shown on Figure 2 and Table 5

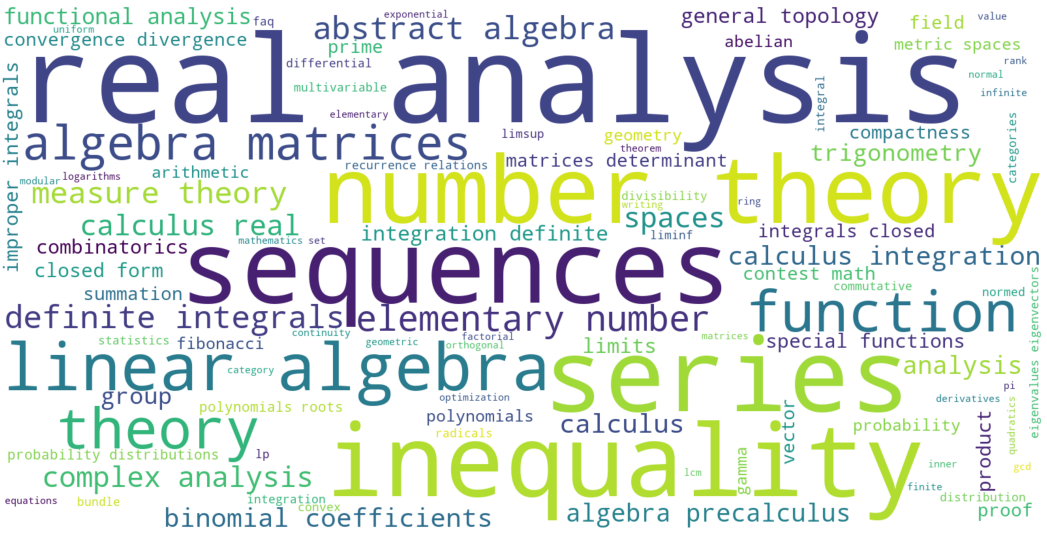

Figure 2: Word cloud visualization of mathematical topic tags for the 200 problems in RHIM. Word size represents tag frequency. The benchmark exhibits substantial diversity across core mathematical domains, with prominent coverage of real analysis, abstract algebra, number theory, sequences, calculus, linear algebra, geometry, and inequalities, alongside specialized topics including measure theory, topology, and combinatorics.

Table 5: Top 20 mathematical topic tags in RHIM.

Topic	**N**	**%**	**Topic**	**N**	**%**
Real analysis	43	21.5	Measure theory	13	6.5
Sequences & series	29	14.5	Complex analysis	12	6.0
Calculus	27	13.5	Binomial coeff.	10	5.0
Inequality	25	12.5	Number theory	9	4.5
Linear algebra	22	11.0	Algebra precalc.	9	4.5
Integration	20	10.0	Trigonometry	8	4.0
Matrices	17	8.5	Functional anal.	8	4.0
Elem. number theory	15	7.5	General topology	7	3.5
Definite integrals	15	7.5	Limits	7	3.5
Abstract algebra	15	7.5	Polynomials	7	3.5

A.8 MODEL HYPER-PARAMETERS

Model hyper-parameters was show in Table 6

Table 6: Generating parameters for various LLMs.

Model	**Generation Setup**
deepseek-chat	`model = deepseek-chat, temperature = 0, max_tokens = None`
deepseek-reasoner	`model = deepseek-reasoner, temperature = 0, max_tokens = None`
Qwen3-Next-80B-A3B-Instruct	`model = Qwen3-Next-80B-A3B-Instruct, temperature = 0, max_tokens = None`
gemini-2.5-flash	`model = gemini-2.5-flash, temperature = 0, max_tokens = None`
gpt-5.2	`model = gpt5.2, temperature = 0, max_tokens = None, resoning=medium`

A.9 Role Specification and Prompt Template

This appendix provides detailed role specifications, responsibilities, and prompt templates for the two pipeline architectures evaluated in section 5 .

A.9.1 Pipeline 1: Solver-Verifier (Streamlined)

Architecture Overview. Pipeline 1 uses a two-agent architecture following the solver-verifier pattern: a single solver agent handles all reasoning tasks (stages 1-5), followed by a separate verifier agent that validates the generated proof (stage 6).

Agent Roles and Responsibilities. Solver Agent (Gemini-2.5-Flash):

- **Stage 1 (Detection):** Determine whether the problem contains a redundant hypothesis
- **Stage 2 (Identification):** Identify which specific hypothesis is redundant
- **Stage 3 (Reformulation):** Create a new proof problem where the redundant hypothesis becomes the target theorem to be proved from the remaining hypotheses
- **Stage 4 (Proof Sketch):** Generate a high-level proof strategy and outline
- **Stage 5 (Detailed Proof):** Write a complete, rigorous mathematical proof following the sketch

Verifier Agent (DeepSeek-Reasoner):

- **Stage 6 (Verification):** Validate the logical correctness of the proof and determine acceptance/rejection

```
you are judge.
goal:
    Read a structured mathematics problem.
    Answer the question Q1: 'Does it problem have a redundant assumption?
    '
    If it has a redundant assumption, create a new problem that we can
    deduce the redundant assumption from the other assumptions.

    You need to write down the answer follow the below format.
    ####BEGIN_OF_FORMAT_PART1###
    Answer to Q1: <yes/no; If the problem has a redundant assumption you
    must write down yes, if the problem does not have a redundant
    assumption you must write down no>

    Redundant assumption: <if the problem has a redundant assumption you
    must write down the redundant assumption, if the problem does not
    have a redundant assumption you must write down "The problem does not
     have a redundant assumption.">

    Assumptions: <if the problem has a redundant assumption you must
    exclude the redundant assumption and write down the other assumptions
     as I give you, else write "no">

    Ordinal number of redundant assumption: <if the problem has a
    redundant assumption you must write down the ordinal number of the
    redundant assumption follow the structure, if the problem does not
    have a redundant assumption you must write down "-1">
    ###END_OF_FORMAT_PART1###
    To prove an assumption is redundant, we usually consider it as
    important result which can be deduced from the other given
    assumptions. Based on that logic, we will construct a new problem
    which have assumptions from the original problem exclude the
    redundant assumption we suppose and the redundant assumption as an
    important result we need to prove it right.
    First, you need to write down the new problem follow the structure
```

```
    ####BEGIN_OF_FORMAT_PART2###
    Assumptions:
    Assumption <number>: <Write the first assumption here>
    Assumption <number>: ...
    ...
    Problem:
    <Write the redundant assumption here>
    ###END_OF_FORMAT_PART2###
    Now we have the new problem. Your task is to prove the new problem.
    ####BEGIN_OF_FORMAT_PART3###
    detailed proof:
    <You need to prove the new problem. You need to prove that you can
    deduce redundant assumption from the others redundant assumption. If
    there is not a new problem you must write down "We don't have a new
    problem so we don't have a detailed proof>

    ###END_OF_FORMAT_PART3###

guidelines: Guideline_1:

general rules:
- think step-by-step, cite what you did.
- be concise and specific.

<Problem_with_redundant_assumption or
    Original_problem_with_numerical_assumption>
```

Listing 4: Pipeline 1 Solver Prompt

```
you are final reviewer.
goal:
        You will read the process another agent detect whether a
    mathematical proof problem has a redundant assumption $A_i$, identify
     that redundancy and try to justify the redundancy by proving the
    redundant assumption is true only by using the other assumption (
    without using redundant assumption).
        Check correctness of the proof; you should output the answer in
    the following format:
        ###BEGIN_OF_FORMAT###
        answer_proof_review:
        proof review: <True/False; If the proof is correct, fill True, if
     the proof is incorrect, you should write down False>
        finished: <yes/no; If there isn't detailed proof, you should
    return yes. If there is detailed proof, you should check the
    correctness of the proof and return the answer. If the proof is
    correct, you should write down yes, if the proof is incorrect, you
    should write down no. >
        clear answer: <yes/no; If the proof is correct, you should write
    down yes, if the proof is incorrect, you should write down no>
        end_of_proof_review
        ###END_OF_FORMAT###

guidelines: Guideline_1:

general rules:
- think step-by-step, cite what you did.
- be concise and specific.

<if detection is "no"
    then FULL_CONTEXT = judge_prompt (include problem) + output_from
    judge
 else
```

```
    FULL_CONTEXT = judge_prompt (include problem) + output_from judge>
```

Listing 5: Pipeline 1 Verifier Prompt

A.9.2 PIPELINE 2: TASK-DECOMPOSITION (MULTI-AGENT)

Architecture Overview. Pipeline 2 decomposes the reasoning task into four specialized agents, each responsible for a focused subtask: a judge handles detection and reformulation (stages 1-3), a proof strategy planner creates proof outlines (stage 4), a mathematician writes formal proofs (stage 5), and a verifier validates correctness (stage 6).

Agent Roles and Responsibilities. Judge Agent (DeepSeek-Chat):

- **Stage 1 (Detection):** Determine whether the problem contains a redundant hypothesis
- **Stage 2 (Identification):** Identify which specific hypothesis is redundant
- **Stage 3 (Reformulation):** Create a proof problem with the redundant hypothesis as target theorem

Proof Strategy Planner Agent (DeepSeek-Chat):

- **Stage 4 (Proof Sketch):** Given the reformulated problem, generate a high-level proof strategy

Mathematician Agent (DeepSeek-Chat):

- **Stage 5 (Detailed Proof):** Given the proof sketch, write a complete rigorous proof

Verifier Agent (DeepSeek-Reasoner):

- **Stage 6 (Verification):** Validate the proof and determine acceptance/rejection

```
you are judge.
goal:
    Read a structured mathematics problem and write answers carefully and
     concisely.
    Answer the question Q1: 'Does the problem have a redundant assumption
    ?'
    If the problem has a redundant assumption.
    You need to write down the answer follow the below format.
    ####BEGIN_OF_FORMAT###
    Answer to Q1: <yes/no; If the problem has a redundant assumption you
    must write down yes, if the problem does not have a redundant
    assumption you must write down no>

    Redundant assumption: <if the problem has a redundant assumption you
    must write down the redundant assumption, if the problem does not
    have a redundant assumption you must write down "The problem does not
     have a redundant assumption.">

    Assumptions: <if the problem has a redundant assumption you must
    exclude the redundant assumption and write down the other assumptions
     as I give you, else write "no">

    Ordinal number of redundant assumption: <if the problem has a
    redundant assumption you must write down the ordinal number of the
    redundant assumption follow the structure, if the problem does not
    have a redundant assumption you must write down "-1">
    ###END_OF_FORMAT###
```

```
guidelines: Guideline_1:

general rules:
- think step-by-step, cite what you did.
- be concise and specific.

<Problem_with_redundant_assumption or
    Original_problem_with_numerical_assumption>
```

Listing 6: Pipeline 2 Judge Prompt

```
you are proof strategy planner.
goal:
    Read a structured mathematics problem and write answers carefully and
     concisely follow the JSON structure or JSON object as mentioned in
    guidelines.
    Now break this mathematic problem into clear, minimal steps and note
    them.
    Use save_note to let mathematician and proof writer read your proof
    sketch.
    Your answer should be in the following format:
    ###BEGIN_OF_FORMAT###
    proof sketch:
    <Write the proof sketch here if you were given a new problem, if you
    were not given a new problem you must write down "We don't have a new
     problem so we don't have a proof sketch">
    end_of_proof_sketch
    ###END_OF_FORMAT###

guidelines: Guideline_1: Guideline_2: Use the following format for your
    proof sketch: Step 1\) ...
Step 2\) ...
Step <number_of_steps>\) ...

general rules:
- think step-by-step, cite what you did.
- be concise and specific.

The new problem is
The problem does not have a redundant assumption. You must write down 'no
    '
```

Listing 7: Pipeline 2 Planner Prompt

```
you are mathematician and proof writer.
goal: Read the new problem and the proof sketch and write a detailed
    proof for those subgoals in proof sketch.
        Your answer should be in the following format:
        ###BEGIN_OF_FORMAT###
        detailed proof:
        <Write the detailed proof here if you were given a new problem
    and proof sketch, if you were not given a proof sketch you must write
     down "We don't have a proof sketch so we don't have a detailed proof
    ">
        end_of_detailed_proof
        ###END_OF_FORMAT###

guidelines: Guideline_1:

general rules:
```

```
- think step-by-step, cite what you did.
- be concise and specific.

The new problem is
The problem does not have a redundant assumption. You must write down 'no
    '
Proof sketch is
We don't have a new problem so we don't have a proof sketch
```

Listing 8: Pipeline 2 Mathematician Prompt

Prompt Template for Verifier Agent. The verifier prompt is identical to Pipeline 1 (see Listing 5).

A.10 PIPELINE COMPARISON

Table 7 summarizes the architectural differences between the two pipelines.

Table 7: Architectural comparison of Pipeline 1 (Solver-Verifier) and Pipeline 2 (Task-Decomposition)

Aspect	**Pipeline 1**	**Pipeline 2**
Pattern	Solver-Verifier	Task-Decomposition
Number of agents	2	4
Stages 1-5 model	Gemini-2.5-Flash	DeepSeek-Chat
Stage 6 model	DeepSeek-Reasoner	DeepSeek-Reasoner
Task distribution	Monolithic (1 agent)	Specialized (3 agents)
Hypothesis	Simple separation sufficient	Fine-grained decomposition improves performance

A.11 IMPLEMENTATION NOTES

Error Handling. Both pipelines implement error handling for:

- JSON parsing failures: Retry with simplified format
- Agent timeouts: Maximum 3 retries with exponential backoff
- Malformed outputs: Request reformatting with explicit instructions

Prompt Engineering Techniques. All prompts incorporate:

- **Structured output format:** JSON schemas enforce consistency
- **Role specification:** Clear identity priming for each agent
- **Task decomposition:** Explicit step-by-step instructions
- **Format examples:** Few-shot examples in actual implementation (omitted here for brevity)

Temperature Settings.

- Judge/Solver agents: Temperature = 0.0 (allow creative problem-solving)
- Planner agent: Temperature = 0.0 (balance creativity and structure)
- Mathematician agent: Temperature = 0.0 (prioritize rigor and precision)
- Verifier agent: Temperature = 0.0 (maximize consistency in verification)