WWW.IJRPR.COM

**International Journal of Research Publication and Reviews**

Journal homepage: www.ijrpr.com ISSN 2582-7421

# VERIFAI – INDEPENDENT VERIFICATION LAYER FOR AI-GENERATED CLAIMS

***Mr. Sanjay Krishna B[1], Mrs. Sabeetha M[2], Arun Prasath P[3], Dhanushree K.L.[4], Durga V.B.[5], Deepak Raj D[6], Daranya G[7]***

Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, Coimbatore – 641062, Tamil Nadu, India

**A B S T R A C T :**

Artificial intelligence (AI) systems are increasingly used to generate information and support decision-making. However, their responses may contain inaccurate, outdated, inconsistent, or hallucinated claims, making independent verification essential. This paper presents VerifAI, an independent verification layer designed to assess the reliability of AI-generated responses. The proposed system analyzes generated claims, checks them against available trusted sources, and produces confidence-based assessments to help users identify potentially unreliable information. It also aims to compare responses from multiple AI models to identify inconsistencies and highlight claims requiring further verification. VerifAI is designed with a modular architecture that can support integration through an application programming interface (API) and a Model Context Protocol (MCP) server. The system aims to improve transparency, support evidence-based evaluation, and reduce reliance on unverified AI-generated content. The implemented workflow is covered by automated regression tests, and its factual accuracy, evidence support, consistency, and verification performance remain to be evaluated with labeled data and live provider accounts. The project provides a foundation for developing a practical and extensible approach to trustworthy AI-assisted information processing.

**Keywords:** VerifAI, Artificial Intelligence, Hallucination Detection, Claim Verification, AI Reliability, Cross-Model Consistency, Trustworthy AI.

## Introduction

In the present digital era, Large Language Models (LLMs) have become part of everyday study, research, software development, and decision making. A student, an engineer, or an analyst can now obtain a fluent, well-structured answer to almost any question within seconds. While this convenience improves productivity, it also introduces a serious challenge: a fluent answer is not necessarily a correct answer. LLMs are known to produce statements that sound confident and authoritative but are factually wrong, unsupported, or partly invented, a behavior widely described as hallucination [1, 2].

Most AI chat applications are designed to generate answers rather than to examine them. A user who receives a single response from a single model has no independent way to know which statements are reliable. To compare answers, users must manually copy the same prompt into several provider websites, lose conversational context, and then compare the results by reading them side by side. This creates what can be called a single-model authority illusion, where persuasive language is mistaken for evidence. Therefore, there is a need for a system that can compare independent model outputs, examine the important factual claims, and present the supporting or opposing evidence in a transparent way [1, 3].

Advances in transformer-based language models have made it possible to generate high-quality text at scale [4, 5]. At the same time, research in fact verification, retrieval-augmented generation, self-consistency, and multi-agent debate has shown that the factual quality of model output can be assessed and improved when claims are decomposed and checked against evidence [6, 7, 8, 9, 10]. These techniques motivate the design of an independent verification layer that sits between the user and AI-generated text.

The project “VerifAI: Independent Verification Layer for AI-Generated Claims” is developed to address these challenges. VerifAI is an AI response verification, multi-model orchestration, and evidence analysis platform. A user can submit one prompt to one, two, or three independently selected language models using their own API keys, inspect each answer in a separate lane, and then verify the factual claims of any completed answer against attributed evidence. Where the user requests it, the platform can also combine the answers into a clearly labeled summary.

In addition, the system introduces design rules that keep the platform honest: every answer is attributed to its exact provider and model, a failed model call is never displayed as a success, a combined summary is always marked UNVERIFIED until it is independently checked, and the platform never manufactures certainty, citations, rankings, or scores. By converting raw model output into structured, evidence-linked information, the system aims to support users in deciding how far an AI-generated answer can be trusted.

## Literature Review

### *1.1. Hallucination and Reliability of LLM-Generated Content*

Large Language Models are trained on very large text corpora and can generate fluent responses for a wide range of tasks. Few-shot learning results showed that a single pretrained model can perform many tasks from a handful of examples, which led to rapid adoption of these models in applications [5]. The underlying Transformer architecture, which relies on attention mechanisms to model relationships between words, is the foundation of this progress [4].

Despite their fluency, language models can generate content that is not faithful to any source or is simply false. A comprehensive survey of hallucination in natural language generation describes intrinsic and extrinsic hallucination, discusses their causes in data, training, and decoding, and highlights that such errors are difficult for users to notice because the text reads naturally [1]. The TruthfulQA benchmark further showed that larger models can still imitate common human falsehoods, so model size alone does not guarantee truthfulness [2].

These findings show that the reliability of AI-generated content cannot be assumed. Therefore, a practical system must treat every model answer as a set of claims that may or may not be supported. The VerifAI system is built on this principle: it never treats a model output as evidence by itself, and it keeps the answer separate from the verification of that answer.

### *1.2. Claim Extraction, Fact Verification and Retrieval*

Automated fact checking is commonly framed as the task of finding evidence and then deciding whether a claim is supported, refuted, or lacks enough information. The FEVER dataset formalized this task at scale by pairing claims with evidence from Wikipedia and labeling them as supported, refuted, or not enough information [6]. This three-way view is important because it recognizes that some claims simply cannot be decided from the available evidence.

Long model answers contain many separate facts, so judging the whole answer as true or false is too coarse. FActScore addresses this by breaking a generation into atomic facts and measuring the proportion that is supported by a knowledge source, which gives a fine-grained estimate of factual precision [7]. Retrieval-augmented generation similarly combines a language model with an external retrieval component so that outputs can be grounded in retrieved documents rather than in model memory alone [8].

The VerifAI verification engine follows this claim-oriented approach. It extracts factual claims from a completed answer, retrieves evidence, and assigns a verdict with confidence and limitations. Unlike a fixed benchmark pipeline, VerifAI also treats evidence quality as something that must be measured independently, and it states openly that domain-calibrated precision still needs separate evaluation.

### *1.3. Multi-Model Comparison, Consistency and LLM-as-Judge*

Several lines of research use more than one generation to improve or assess reliability. Self-consistency samples multiple reasoning paths and selects the answer that is most consistent, which improves accuracy on reasoning tasks [9]. SelfCheckGPT detects hallucinations by sampling several responses and checking whether the statements in an answer are consistent with the samples, without needing any external database [11]. Multi-agent debate lets several model instances propose and critique answers over rounds, and has been reported to improve factuality and reasoning [10].

Another important idea is the use of a strong language model as an evaluator. The MT-Bench and Chatbot Arena study showed that LLM judges can agree with human preferences at a high rate, while also documenting biases such as position bias and verbosity bias [3]. These results justify the use of model-based judging in verification, but they also warn that a judge model is itself fallible and must be combined with rule checks and evidence.

The VerifAI system uses these ideas carefully. It compares independent outputs from different models, but it does not claim that agreement proves correctness, because several models can share the same mistake. Agreement and disagreement are treated as signals to investigate rather than as verdicts, and the platform explicitly notes that its current cross-model disagreement measure is verdict-based and not yet a full claim-level conflict matrix.

### *1.4. Benchmarks, Leaderboards and Evaluation*

Public benchmarks and leaderboards help users compare models. Chatbot Arena collects human preference votes in an open platform and ranks models from pairwise comparisons [12]. LiveBench was designed as a challenging benchmark that limits test-set contamination by using frequently updated questions with objective ground-truth scoring [13]. TruthfulQA targets truthfulness specifically [2].

However, a leaderboard measures model performance on a benchmark and not the reliability of one particular answer to a user's question. VerifAI therefore keeps these ideas separate. Its public leaderboard shows a dated, source-attributed snapshot derived from LiveBench, while its own VerifAI leaderboard is deliberately left empty until valid first-party labeled evaluations exist. This design avoids presenting fabricated scores as if they were measurements.

### *1.5. Existing Systems and Their Limitations*

Several categories of existing tools address parts of this problem. General AI chat applications focus on answer generation from a single model and usually do not provide side-by-side comparison, claim-level checking, or a record of provenance. Answer engines such as Perplexity and research assistants such as Elicit, Consensus, and Scite emphasize retrieval, citation, and synthesis of literature, while public leaderboards such as Arena and LiveBench rank models in general rather than checking a specific answer.

A major limitation of these categories is that they are separated from one another. A user who wants to compare several models, check the factual claims of the result, and keep an audit trail must combine several disconnected tools by hand. Many tools also present synthesis as a polished final answer, which can hide contradictions between sources or models, and they rarely state plainly when a claim could not be verified.

Another limitation concerns credentials and safety. Applications that call model providers on behalf of many users may rely on shared keys, and prompts may contain secrets that should never leave the user's environment [14]. The VerifAI system addresses these gaps by connecting model choice, independent generation, claim decomposition, evidence analysis, disagreement exposure, and a persistent decision trail in a single BYOK workspace, while openly documenting what is still incomplete.

Taken together, the surveyed research and tools leave a clear gap. Benchmarks rank models in general, retrieval assistants concentrate on literature synthesis, and chat applications concentrate on generation. None of them connects independent multi-model answers, claim-level evidence checking, and a persistent record for one user's specific question. VerifAI is therefore positioned as a verification workspace rather than as a new model or benchmark, and its contribution is the connected workflow together with the honest reporting of what remains unverified.

## Proposed System and Methodology

**Table 1 - Main modules of the VerifAI system.**

| Module | Responsibility |
|---|---|
| Multi-model orchestration | Sends prompts to selected AI models and records each model response independently. |
| Claim extraction | Identifies individual factual claims in a generated response for checking. |
| Evidence retrieval | Retrieves relevant evidence from available sources and the private knowledge base. |
| Claim verification | Assesses claims against evidence and records verdicts with supporting rationale. |
| Confidence and uncertainty | Communicates confidence and limitations rather than presenting unsupported certainty. |
| History, reports and dashboard | Stores verification history and presents results and benchmark information. |

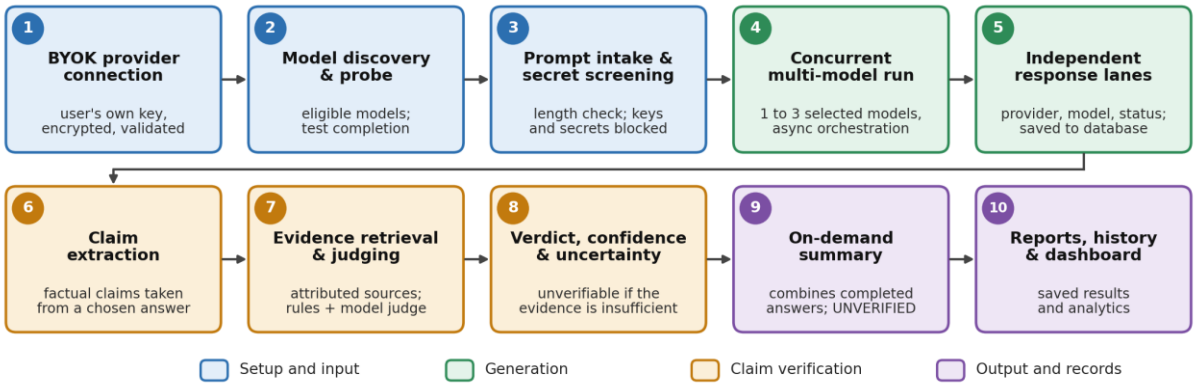



**Fig. 1 - End-to-end workflow of the VerifAI system, from BYOK provider connection to reports and dashboard.**

### *1.6. BYOK Provider Connection and Initialization*

The proposed system begins with the user connecting an AI provider using a key that the user owns. Supported connection paths include Groq, OpenRouter, NVIDIA NIM, and compatible hosted or custom endpoints, together with native adapters for other vendor families. The key is sent to the backend, stored encrypted, and associated only with the owner's account.

The backend validates the connection before it can be used for chat. Only supported and validated BYOK connections can execute requests, and global environment keys are deliberately disabled for owner-specific chat. This guarantees that a missing connection results in a clear failure instead of silently using someone else's credential.

### *1.7. Model Discovery and Completion Probing*

Once a provider is connected, the system discovers the provider-specific models that are eligible for chat through the model discovery endpoint. OpenRouter chat is restricted to explicitly free routes in the current implementation, and the interface offers search and a free-route filter over the discovered catalog.

Catalog presence alone is not accepted as proof. A candidate model is tested through a probe request, and it is marked working only after a real, non-empty completion succeeds. The Find working models action probes candidates and selects up to three successful ones, while known failed models and providers are deprioritized or skipped.

### *1.8. Prompt Intake and Credential Safety Screening*

The user enters a prompt of up to 20,000 characters and chooses one to three distinct models. The backend rejects duplicate selections and verifies that the conversation belongs to the current user. Bounded recent context of up to six previous messages is added, truncated, and labeled as context instead of system instructions.

Before any outbound inference, the prompt and the prior context are screened for recognizable API keys, passwords, bearer tokens, private keys, and connection secrets. A matching request is blocked. This detection is pattern-based and is described as defense in depth rather than a complete data-loss-prevention guarantee [14].

### *1.9. Concurrent Multi-Model Orchestration*

The chat service calls the MultiLLMOrchestrator, which retrieves the owner's credentials, registers the matching provider adapters, and runs all selected models concurrently using asynchronous execution. Native adapters handle Anthropic and Gemini-style protocols, and OpenAI-compatible adapters handle hosted and custom compatible endpoints.

Because the calls run at the same time, the total waiting time is governed mainly by the slowest model rather than by the sum of all models. The standard request still waits for the orchestration result before showing answers, and progressive per-model delivery is recorded as future work.

### *1.10. Independent Response Lanes and Persistence*

Each model returns its own result containing the provider, model name, response, completion status, latency, and failure metadata. The interface shows every model in a separate lane, so that answers are attributed accurately and cannot be mixed. A failed lane shows its error and is never counted as a successful answer.

The user message, the model runs, and their outcomes are stored in PostgreSQL under the owner's conversation. Generated outputs and provider error messages are redacted before they are stored or reused. When the conversation is reopened, the persisted messages and runs are retrieved, so prompt history survives a page reload.

### *1.11. Claim Extraction*

When the user selects Verify this response, a completed answer is passed to the verification workflow. The verification engine extracts the factual claims contained in the answer, so that the assessment is made on individual statements rather than on the whole text at once.

Claim-level assessment is important because a long answer usually mixes correct, incorrect, and unverifiable statements. Working at the claim level follows the direction of fact-verification and factual-precision research and allows the user to see exactly which statement is weak [6, 7].

### *1.12. Evidence Retrieval and Judging*

For the important claims, the verification engine retrieves evidence and records its source. Model-based judging and rule-based checks then assess whether the evidence supports, opposes, or fails to settle each claim. Model judges are known to be useful but fallible, so their output is combined with rules and retained evidence instead of being accepted blindly [3].

The private knowledge base can supply owner-isolated documents as an additional evidence source. The quality of evidence and the precision of verdicts in specific domains require independent measurement, and the platform states this limitation instead of hiding it.

### *1.13. Verdicts, Confidence and Uncertainty*

Each claim receives a verdict together with a confidence level and a statement of limitations. The system is designed so that a claim can honestly be reported as unverifiable when the evidence is insufficient. It does not manufacture certainty, citations, or scores.

This approach replaces a simple true or false label with an evidence-based assessment. Users can see the reasoning, the supporting and opposing evidence, and what remains unresolved before deciding how to use the answer [6].

### *1.14. On-Demand Summarization and Disagreement Analysis*

A combined answer is created only when the user clicks Summarize. Only completed, non-empty saved responses are included. The summary prompt asks the model to combine complementary information, identify disagreement and uncertainty, avoid inventing facts, and treat the source responses as data and not as instructions.

The service tries available models that answered successfully, using the owner's own credential. A successful synthesis is saved and reused, and the original independent answers remain available. The summary is always labeled UNVERIFIED, and if every attempt fails the API reports an error rather than inventing a summary. The present disagreement flag is verdict-based, and a claim-level agreement and conflict matrix is a planned improvement.

### *1.15. Reports, History and Dashboard Deployment*

The final output is delivered through the web interface. Conversations, runs, and verification records are available from the History page, and the Reports and Dashboard pages present analytics based on persisted data. The Knowledge Base, the Leaderboard, and the browser extension pathway complete the workspace.

The Leaderboard shows a dated, source-attributed public benchmark snapshot, and the first-party VerifAI leaderboard stays empty until valid labeled evaluations exist. This user-friendly frontend ensures that both technical and non-technical users can compare, verify, and review AI answers with a clear view of what has and has not been established.

## Implementation

### *1.16. Frontend*

The frontend of the VerifAI system is designed to provide a clear, responsive, and trustworthy interface for comparing and verifying AI answers. It is developed using React and TypeScript with Vite as the build tool, and React Router is used for navigation between the main pages. The frontend serves as the primary layer through which users connect providers, select models, send prompts, read results, and open verification records [15].

The system uses React as the core framework for building the interface. Its component-based structure allows reusable elements such as model lanes, provider cards, status badges, verification panels, and history lists. This modular structure improves maintainability and makes it easier to add new provider or result views without rewriting existing screens.

The main screen is the multi-model chat page. It includes provider connection controls, a searchable catalog of discovered models, a free-route filter, model probing, an automatic Find working models action, and three selected-model slots. After a prompt is sent, each model appears in its own lane with its status, response, latency, and error details, so that the user can compare answers without mixing them up.

Beside each completed set of answers, the interface offers two clearly separate actions: Summarize, which creates a combined answer on request and labels it UNVERIFIED, and Verify this response, which links to the verification workflow. Additional pages such as History, Reports, Dashboard, Knowledge Base, and Leaderboard present persisted data, and the interface supports both light and dark themes.

### *1.17. Backend*

The backend of the VerifAI system is responsible for credential handling, model orchestration, verification, persistence, and communication between the frontend and external providers. It is developed using Python and FastAPI, which provide high performance and support asynchronous operations. The backend is organized as a modular monolith, so that chat, providers, language-model orchestration, verification, benchmarks, and reports remain separate modules inside one deployable application [16].

For a chat request, the backend first validates conversation ownership and the selected models, and then screens the prompt and the bounded prior context for recognizable secrets. It retrieves only the current owner's validated provider credentials, decrypts them on the server, and registers the matching adapter for that request. Global environment keys are intentionally disabled for owner-specific chat, so a missing connection can never silently fall back to a shared key.

The MultiLLMOrchestrator then runs the selected models concurrently using asyncio.gather. Each result carries the provider, model, response, completion status, latency, and failure metadata. A failed model does not stop the others, and a failed lane is never counted as a successful answer. Messages and model runs are stored in PostgreSQL through asynchronous SQLAlchemy, with Alembic managing schema changes [17, 18].

Summarization and verification are separate services. The summary service combines only completed, non-empty saved responses by asking an available responding model, using the owner's own credential, and the saved synthesis is reused on repeated requests. The verification service extracts claims, retrieves evidence, and produces verdicts. Provider responses are treated as untrusted content during synthesis, and generated outputs and provider errors are redacted before storage.

The backend also provides rate limits, owner-scoped queries, protection against unsafe custom endpoints, and a Server-Sent Events route that publishes model lifecycle events. At present, those events are emitted after processing completes and are not token-by-token streaming. Overall, the backend plays a crucial role in turning raw model calls into attributed, persisted, and verifiable results.

### *1.18. Hardware Specification*

The hardware requirements for the VerifAI system are modest, because the heavy language-model inference is performed by external providers using the user's own API keys. The application can therefore be developed and run on standard computing devices, which makes it accessible and cost-effective for students and small teams.

A minimum configuration includes a computer with an Intel Core i5 processor or equivalent, 8 GB of RAM, and at least 256 GB of storage. This is sufficient for running the React frontend, the FastAPI backend, and a local PostgreSQL database in a development environment and for handling moderate workloads.

For improved performance, a higher configuration with a multi-core processor, 16 GB or more RAM, and SSD storage is recommended. These specifications give faster builds, quicker database operations, and smoother handling of the automated test suites. A graphics processor is not required for the current design.

### *1.19. Software Specification*

The VerifAI system is built using a combination of modern software technologies that support web development, asynchronous processing, secure credential handling, and automated testing. The software stack was selected to give high reliability, clear module boundaries, and ease of development.

The backend is developed using Python, with FastAPI for the REST APIs and asynchronous SQLAlchemy for database access. PostgreSQL is the primary database, and Alembic is used to version the schema. Provider integration is handled through native adapters and OpenAI-compatible adapters, which allows both hosted providers and custom compatible endpoints to be used.

The frontend is developed using React with TypeScript and Vite, with React Router for navigation and CSS for responsive layouts and light and dark themes. TypeScript types shared between the API clients and the pages reduce integration errors between frontend and backend.

Security-related software includes Firebase and Google-based authentication, encrypted storage of provider credentials, credential pattern detection and redaction, rate limiting, and SSRF protection. Quality is supported by backend regression suites, TypeScript and build checks, browser regression tests, and full-stack smoke tests executed in GitHub Actions against disposable infrastructure.

**Table 2 - Software technologies used in VerifAI.**

| Layer | Technology | Purpose |
|---|---|---|
| Frontend | React, TypeScript, Vite | Builds the interactive web interface. |
| Backend/API | Python, FastAPI | Provides backend services and REST endpoints. |
| Database access | PostgreSQL, SQLAlchemy | Persists conversations, model runs and verification records. |
| Asynchronous orchestration | Python asyncio | Runs selected model requests concurrently. |
| Authentication and credentials | Firebase/Google authentication; user-owned provider keys | Supports sign-in and the bring-your-own-key workflow. |
| Version control | Git and GitHub | Tracks source-code changes and collaboration. |

## Results and Discussion

### *1.20. Result*

The implementation of VerifAI: Independent Verification Layer for AI-Generated Claims demonstrates the ability to connect user-owned model providers, run one prompt across several models, display independently attributed answers, and route completed answers into a claim-oriented verification workflow. The system was developed to overcome the limitations of single-model chat tools by providing comparison, evidence analysis, and a persistent record instead of a single unchecked answer. This section reports what the inspected implementation and its automated tests show, and it separates implemented features from features that are still planned.

The multi-model chat module is implemented in the source code. A user can connect a provider with a key, discover eligible models, probe a model with a real completion, select up to three distinct models, and send a prompt of up to 20,000 characters. The backend validates ownership, screens the prompt for secrets, retrieves only the owner's credentials, and executes the selected models concurrently through the MultiLLMOrchestrator. Duplicate selections are rejected.

Each model result is returned with its provider, model name, response, completion status, latency, and failure metadata, and the interface shows it in a dedicated lane. Failed calls are shown as failures and are not counted as answers. The conversation, messages, and model runs are stored in PostgreSQL, so that a reopened conversation shows the persisted history together with any saved synthesis.

The on-demand Summarize action is implemented and produces a single combined answer from completed responses by using an available responding model with the owner's credential. The summary is saved, reused on repeated requests, and explicitly labeled UNVERIFIED. When every synthesis attempt fails, the API reports an error and does not invent a summary. The independent answers always remain available.

The verification module is implemented with claim extraction, evidence retrieval, model-based and rule-based judging, verdicts, and linked results that appear in the history and report views. A separate Verify this response action in the chat interface sends a completed answer to this workflow. The precision of evidence retrieval and the calibration of verdicts in particular domains still need independent measurement, and the project does not claim them as achieved.

The supporting modules were also implemented. The frontend provides Chat, Verify, Verification Result, Dashboard, Leaderboard, History, Reports, Knowledge Base, Settings, and landing pages in React and TypeScript with light and dark themes. The public leaderboard presents a LiveBench-derived snapshot, which in the inspected local data covers 52 models with category filters and a weekly refresh workflow [13]. The first-party VerifAI leaderboard intentionally returns an empty list until valid labeled evaluations exist.

Automated testing covers the main behaviors. The backend test suites include checks for model discovery, credential isolation, probing, sensitive-input protection, concurrent execution, synthesis, conversation lifecycle, one to three model validation, SSE lifecycle events, redaction, NVIDIA and custom endpoint scenarios, and multi-vendor on-demand summaries. Development notes record that 1,169 backend tests passed in an isolated environment on 23 October 2026, and a continuous integration workflow runs TypeScript and build checks, browser regressions, backend tests against disposable PostgreSQL, and full-stack smoke tests.

These test results must be read carefully. They are a historical report from development notes and use isolated or mock infrastructure, so they show that the code paths behave as designed but do not prove that every real provider account and model works at this moment. A live end-to-end smoke test with real provider keys and quotas was not part of this evaluation.

**Table 3 - Evaluation areas reported for the current implementation.**

| Evaluation area | What is assessed | Interpretation |
|---|---|---|
| Model orchestration | Model discovery, provider calls, completion status and failure handling. | Functional behavior is tested; results depend on provider and test conditions. |
| Verification workflow | Claim extraction, evidence retrieval, judging and verdict presentation. | Supports claim-level review; evidence quality affects reliability. |
| Credential isolation | Owner-scoped credentials and data access. | Covered by credential-isolation tests on isolated infrastructure; live multi-user validation is pending. |
| Limitations | Streaming, multi-output automatic verification and real-user evaluation. | These remain areas for further development or validation. |

### *1.21. Output Screens and Observed Results*

This subsection presents screenshots captured from the running VerifAI workspace to show the implemented workflow, from provider connection and multi-model chat to claim-level verification, benchmark display and reporting. They are single-session observations of the working system and are not a benchmark of model accuracy. Figs. 2 to 8 present the screens themselves.

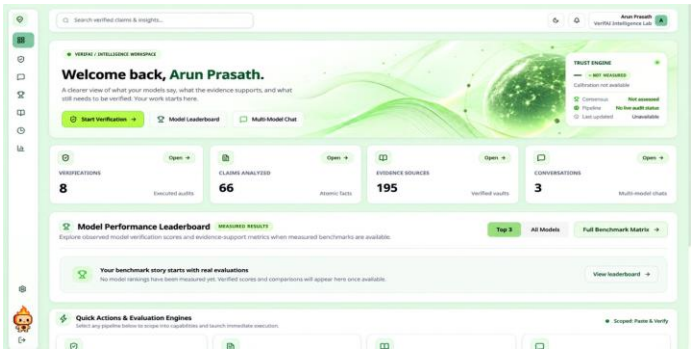



**Fig. 2 - VerifAI dashboard showing verification, claim, evidence-source and conversation counts.**

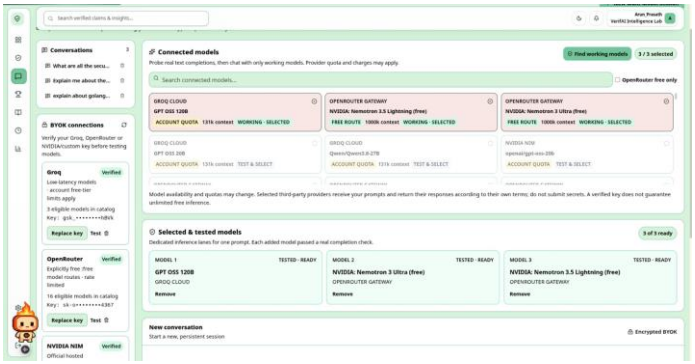



**Fig. 3 - Multi-model chat workspace with verified BYOK connections and three tested models selected.**

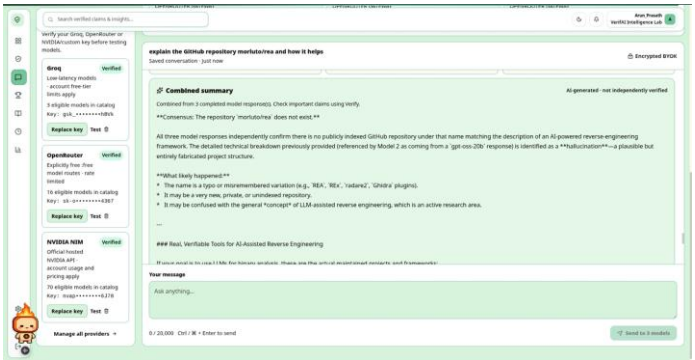



**Fig. 4 - Combined summary of three model responses, labeled as AI-generated and not independently verified.**

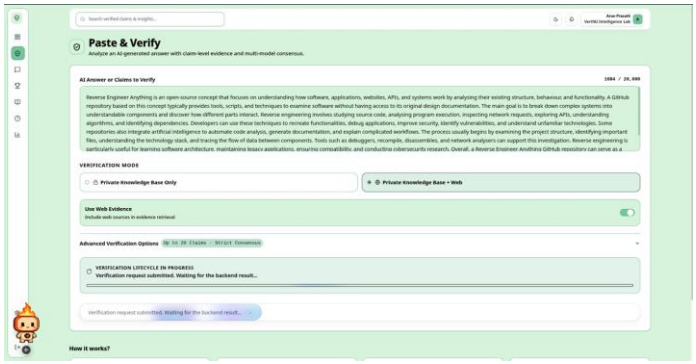



**Fig. 5 - Paste & Verify page with the verification lifecycle in progress.**

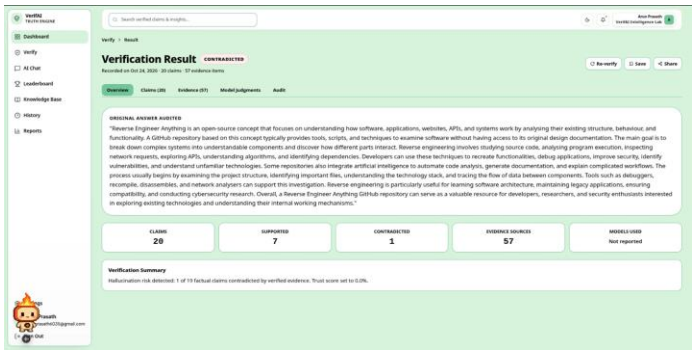



**Fig. 6 - Verification Result page showing the CONTRADICTED status, claim counts and verification summary.**

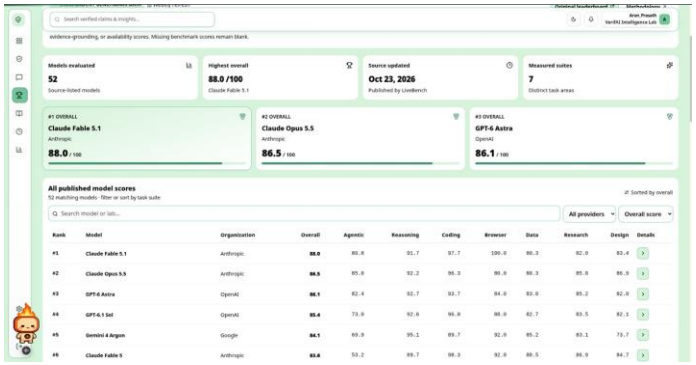



**Fig. 7 - Public leaderboard snapshot derived from LiveBench with per-suite model scores.**

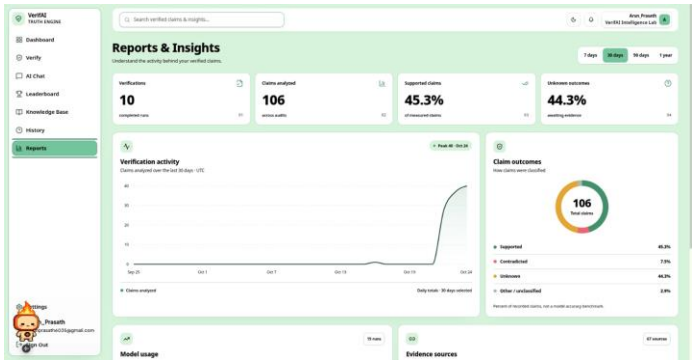



**Fig. 8 - Reports & Insights page summarizing verification activity and claim outcomes over 30 days.**

Overall, the results confirm that VerifAI is more than a chat interface. It is a working, tested multi-model orchestration and verification workspace whose design keeps attribution, failure, and uncertainty visible. At the same time, several capabilities are explicitly recorded as incomplete, and they are discussed in the next section.

### *1.22. Discussion*

The development of VerifAI: Independent Verification Layer for AI-Generated Claims demonstrates the growing need for tools that help people judge AI output instead of simply receiving it. Generative models are now used for tasks where a wrong statement has a real cost, yet the interface of most tools presents every answer with the same confidence. The project addresses this gap by introducing a structured workflow in which generation, verification, and decision support are separate steps [1, 2].

The BYOK design proved to be an important architectural decision. Because the backend retrieves only the current owner's validated credential and does not fall back to a shared key, the platform avoids a class of privacy and cost problems. The decision also means that results depend on each user's provider account, quota, and permissions, and the catalog listing of a model is not proof that a request will succeed. The explicit completion probe was added for this reason.

The honest handling of failure turned out to be a design strength. A lane that truthfully reports a rate limit or an unavailable model is more useful than a counterfeit success, because the user immediately knows which answers can be compared. Similarly, a summary that exposes disagreement is more trustworthy than a polished text that hides it, which is why the combined answer is on demand and always labeled UNVERIFIED.

The discussion also shows that agreement between models should not be treated as proof of correctness. Research on self-consistency and multi-agent debate shows that comparing several generations can improve factuality, but several models can still share a mistake [9, 10]. VerifAI therefore uses disagreement as a signal for investigation and uses evidence, not voting, as the basis of a verdict.

Model-based judging is useful but imperfect. Studies of LLM-as-a-judge report high agreement with human preferences but also biases such as position and verbosity bias [3]. For this reason the verification engine combines model judgment with rule checks and retained evidence and openly states that evidence quality and domain-calibrated precision still require independent measurement.

Several limitations were identified in the current implementation. First, the standard multi-model request is concurrent but is not genuinely token-streamed. The SSE endpoint publishes model-completed and model-failed lifecycle events only after processing has finished, so it does not deliver per-model results as soon as they are ready. True provider token streaming and progressive delivery are planned improvements.

Second, the automatic verification branch in the chat service runs only when a single model result is produced. A three-model request is therefore not verified automatically for every model, although the user can verify a response manually from the interface. As a consequence, the cross-model disagreement flag, which depends on verification verdicts, is not a reliable three-model comparison signal at present, and a claim-level agreement and conflict matrix is needed.

Third, model health is checked by a probe at selection time, but a model that worked earlier can become unavailable later. An expiring health cache, a preflight refresh, classified retries with backoff, and circuit breakers are needed so that the system can recover gracefully. Provider quotas, deprecations, and account permissions can change at any time and are outside the system's control.

Fourth, the credential safety layer is pattern-based. It blocks recognizable secrets and redacts outputs, but it is not a complete data-loss-prevention solution, and it has not yet been tested against a full set of prompt-injection evaluations. Provider responses are treated as untrusted during synthesis, but defense in depth must continue to be strengthened [14].

Fifth, some behaviors described in earlier design notes, such as automatic synthesis, critique and revision, and verification, do not yet match the current chat processing path. For this reason, the source code is treated as the authority for current behavior, and the terms implemented, validated, and generally available are kept distinct and never used interchangeably.

## Conclusion and Future Work

### *1.23. Conclusion*

The project VerifAI: Independent Verification Layer for AI-Generated Claims was developed with the objective of helping people judge AI-generated answers instead of trusting them blindly. In the modern digital era, LLMs produce fluent answers at great speed, but fluency can hide factual errors. Traditional chat tools present a single answer with no independent check, which forces users to compare models and verify claims by hand. The proposed system addresses this challenge by providing a connected workflow for comparison, verification, and decision support [1, 2].

The platform integrates BYOK provider connections, model discovery and probing, concurrent multi-model orchestration, claim-oriented verification, and persistent history into a single workspace. Users can send one prompt to one, two, or three distinct models, read each answer in its own attributed lane, and send completed answers to the verification workflow. This makes the platform useful for students, researchers, analysts, and developers who need to know how far an AI answer can be trusted.

One of the most important achievements of the project is the honest design of the system. Failed calls are never shown as successful, summaries are on demand and labeled UNVERIFIED, credentials are screened and isolated, and the benchmark views state their source or remain empty when no evidence exists. These rules keep attribution, failure, and uncertainty visible and prevent the platform from manufacturing certainty.

The backend, built with FastAPI, asynchronous SQLAlchemy, and PostgreSQL, provides concurrent orchestration, ownership checks, and persistence, while the React and TypeScript frontend provides a clear interface with light and dark themes. Automated regression suites and a continuous integration workflow support the correctness of the implemented code paths, although they do not replace live testing with real provider accounts.

The project also records its limitations openly. True token streaming, automatic verification of all three model outputs, a claim-level disagreement matrix, health-cache expiry with retries, and a first-party reliability evaluation are not yet complete. These gaps do not reduce the value of the work. They define a realistic and measurable path for improvement.

In conclusion, VerifAI is more than a chat interface. It is an evidence-first workspace that supports comparing, verifying, and refining AI answers across models, and it establishes a strong foundation for a trustworthy independent verification layer for AI-generated claims.

### *1.24. Future Enhancement*

Although the current implementation of VerifAI provides a working and tested core workflow, there are several opportunities for future enhancement that can increase its reliability, evidence quality, and industry relevance. The present system is a strong foundation for a larger verification and decision-support platform.

One major future enhancement is progressive delivery and true streaming. Currently the standard request waits for all models, and the SSE route emits lifecycle events after processing. Future versions can stream tokens from the providers and push each model's result as soon as it completes, so that users can start reading the fastest answer while the others are still running.

Verification of every model output is another important enhancement. The automatic verification path should be extended from single-model requests to all selected models, with independent claim verification for each answer and for the final synthesis. This would remove the present limitation in which a three-model run is not automatically verified.

A claim-level agreement and conflict matrix would improve disagreement analysis. Instead of a verdict-based flag, the system can align the claims made by different models, show which claims are shared, which conflict, and which appear in only one answer, and link each cell to its evidence. This would make cross-model comparison much more informative.

Provider reliability can be improved through an expiring model-health cache, preflight refresh, classified retries with backoff, circuit breakers, and graceful replacement of a failing model. Provider trace identifiers, token and cost reporting, and success-rate dashboards would also give users better observability of what each request cost and how reliable each provider has been.

Evidence quality should be strengthened through better retrieval, relevance scoring, opposing-evidence search, and calibration of confidence in different domains. Supported third-party web and scholarly sources must be chosen with legal and commercial considerations in mind. The system should also publish a reproducible internal evaluation method and, only when sufficient ground-truth data exists, a first-party reliability leaderboard.

Synthesis can be extended with an optional structured critique and revision step with bounded cost, and with an editor-model selection that is independently tested. The synthesis itself should be verifiable, so that the final combined answer receives the same claim-level checks as the individual answers.

Security and data governance can be strengthened further through defense-in-depth data-loss prevention, prompt-injection evaluations, organization-level policies, audit logs, configurable retention, and optional separate encryption of saved prompts. An external security review should be completed before the platform is described as enterprise-ready.

The user experience can be improved by cancellation and resumable requests, idempotency for retries, improved mobile and accessibility states, shareable and exportable verification reports, and a deeper integration of the Chrome extension so that AI answers can be verified directly where they are read. Team features such as shared workspaces and review workflows can be explored after the core loop is proven.

Finally, the project should be validated with real users and live provider accounts. A controlled comparison against manual multi-tab workflows, together with measured time savings and reduced false confidence, would show whether the platform delivers its intended value.

## REFERENCES


[1] Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., et al. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys, 55*(12), Article 248, 1–38.

[2] Lin, S., Hilton, J., & Evans, O. (2022). TruthfulQA: Measuring how models mimic human falsehoods. *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL 2022)*, 3214–3252.

[3] Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., et al. (2023). Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *Advances in Neural Information Processing Systems 36 (NeurIPS 2023), Datasets and Benchmarks Track*.

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., et al. (2017). Attention is all you need. *Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS 2017)*, 5998–6008.

[5] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. (2020). Language models are few-shot learners. *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 1877–1901.

[6] Thorne, J., Vlachos, A., Christodoulopoulos, C., & Mittal, A. (2018). FEVER: A large-scale dataset for fact extraction and verification. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT 2018)*, 809–819.

[7] Min, S., Krishna, K., Lyu, X., Lewis, M., Yih, W.-t., et al. (2023). FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023)*, 12076–12100.

[8] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 9459–9474.

[9] Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., et al. (2023). Self-consistency improves chain of thought reasoning in language models. *Proceedings of the 11th International Conference on Learning Representations (ICLR 2023)*.

[10] Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., & Mordatch, I. (2024). Improving factuality and reasoning in language models through multiagent debate. *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, PMLR Vol. 235.

[11] Manakul, P., Liusie, A., & Gales, M. J. F. (2023). SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023)*, 9004–9017.

[12] Chiang, W.-L., Zheng, L., Sheng, Y., Angelopoulos, A. N., Li, T., et al. (2024). Chatbot Arena: An open platform for evaluating LLMs by human preference. *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, PMLR Vol. 235.

[13] White, C., Dooley, S., Roberts, M., Pal, A., Feuer, B., et al. (2025). LiveBench: A challenging, contamination-limited LLM benchmark. *Proceedings of the 13th International Conference on Learning Representations (ICLR 2025)*.

[14] OWASP Foundation. (2023). *OWASP Top 10 for large language model applications, version 1.1*. OWASP Foundation Project Documentation.

[15] React. (2024). React: The library for web and native user interfaces. *Official React Documentation*.

[16] FastAPI. (2024). FastAPI framework guide for high-performance API development. *Official FastAPI Documentation*.

[17] PostgreSQL Global Development Group. (2024). *PostgreSQL documentation*. Official PostgreSQL Documentation.
[18] SQLAlchemy. (2024). Asynchronous I/O (asyncio) extension. *Official SQLAlchemy Documentation*.