Bridging Human Verification and Automated Fact-Checking: A Review of Current AI Chatbot Testing Frameworks

Meshaal Al-Saffar   
*Dept. of Computer Science, College of Engineering*   
*Qatar University*Doha, Qatar  
[200607511@qu.edu.qa](mailto:200607511@qu.edu.qa)

Abdelkarim Erradi  
*Dept. of Computer Science, College of Engineering*   
*Qatar University*Doha, Qatar  
[erradi@qu.edu.qa](mailto:erradi@qu.edu.qa)

*Abstract* — Factual accuracy remains a stubborn weakness of web-deployed AI chatbots, yet existing evaluation tools each solve only part of the problem. Truth-centered benchmarks rate answers but not fully address long-term robustness and reproducibility. This review analyzes nine peer-reviewed frameworks (TruthfulQA, ChatEval, FACT-GPT, ChatClimate, MutaBot, BoTest, DialTest, Bottester and OggyBug) across eight dimensions covering knowledge bases, scoring scales, human involvement, software stacks and testing methodology. The review reveals three gaps: (i) no confidence-based workflow that summons human experts only when automated verdicts are uncertain, (ii) no durable repository that tracks test cases and results over time, and (iii) no containerized setup for replaying evaluations across model iterations. To bridge these gaps, we propose FactBridgeAI, a Docker-orchestrated, semi-automated human-in-the-loop framework that stores versioned tests in a relational database, records every run for analysis, and employs an LLM verifier that escalates low-confidence cases to expert reviewers. The design combines the speed and scale of automated scoring with the depth, selectivity and auditability of human judgement.

Keywords — Conversational Agent, Chatbot, Automated Testing, Artificial Intelligence

# Introduction

AI chatbots are no longer standalone features, they are embedded directly into modern web applications as JavaScript widgets, iframes, or conversational micro-frontends. From a web development perspective, they behave just like any other interactive component: they receive user input, call backend APIs, display dynamic content, and are deployed via the same pipelines as the rest of the app. As chatbots become central to user experience, handling support, navigation, and transactions, the truthfulness of their responses becomes a critical quality dimension, just like performance, accessibility, and security. Fact-checking AI chatbots is thus a natural extension of web development, especially when implemented as a Dockerized REST microservice that logs results to a database and integrates with CI/CD pipelines. It helps developers ensure that user-facing conversational content aligns with the web app’s static content, maintains trust, and avoids misinformation risks. As conversational UIs increasingly replace traditional forms and menus, verifying chatbot accuracy becomes essential to building reliable and responsible web interfaces.

AI-powered chatbots now act as the first touch-point for banking customers, hospital patients and search-engine users because large language models (LLMs) can craft fluent, context-aware replies in real time. Their persuasive fluency, however, comes with a critical liability: LLMs can “hallucinate,” producing incorrect statements that affect user trust, trigger costly misinformation cascades and, in regulated domains such as finance or healthcare, create legal and safety risks. Ensuring factual accuracy therefore ranks among the most pressing quality-assurance problems facing web-deployed conversational systems.

Researchers have tackled slices of this problem through specialized testing tools. TruthfulQA [1] probes models with misconception-rich questions to reveal falsehood mimicry; ChatEval [2] crowdsources user judgements on dialogue quality; FACT-GPT [4] matches new claims to previously fact-checked ones; ChatClimate [5] grounds answers in climate-science corpora; and frameworks such as MutaBot [3], BoTest [6], DialTest [7], Bottester [8] and OggyBug [9] stress-test robustness via mutation, paraphrase or scripted scenarios. Yet three overarching challenges remain. **First, reproducibility:** none of the existing tools stores test artifacts and results in a durable, queryable repository for long-term analysis. **Second, execution environment:** few provide a containerized execution environment for maintainability and scalability. **Third, cost-effective human oversight:** current pipelines either rely on expensive manual grading or apply fully automated scoring without a mechanism to summon experts only when automated confidence is low.

To clarify how far the field has progressed and where critical gaps persist, this review analyzes the nine frameworks listed above across eight dimensions: (1) their knowledge bases, (2) evaluation approaches, (3) points of human involvement, (4) truth-scoring scales, (5) who assigns the score, (6) back-end stacks and (7) front-end stacks, and (8) the testing methodologies they embody. The latter are grouped, following Lin et al. [10], into (a) usability testing, (b) simulated-user testing, (c) AI-planning testing, (d) metamorphic testing and (e) paraphrase testing.

Responding to these gaps, we outline **FactBridgeAI**, a semi-automated, human-in-the-loop fact-checking framework that (i) stores test cases and outcomes in a relational database for long-term analysis, (ii) orchestrates both the chatbot under test and the evaluation harness within Docker containers for reproducible re-execution, and (iii) integrates an LLM-based verifier that escalates only low-confidence cases to human experts, balancing speed with authoritative judgement.

# Background

*Chatbots* are software agents that conduct text or voice conversations with users, usually through a web or mobile interface. Modern systems employ *large language models* (LLMs), neural networks trained on billions of tokens, to generate context-aware replies. Because LLMs predict the most likely next word, they can produce *hallucinations*: fluent statements that have no factual basis in the training data. Mitigating hallucinations is now central to quality assurance for conversational systems.

A *claim* is a factual proposition expressed by a user or a chatbot (e.g., “The Amazon River is the world’s longest river”). *Fact-checking* evaluates that claim against a *knowledge base*, a trusted corpus such as Wikipedia or a domain-specific database, and assigns a *verdict* (e.g., *CORRECT*, *SUPPORTS* or *NOT ENOUGH INFO*). A *truth-scoring scale* can be binary, ordinal (Likert), or continuous (0–1 confidence). In *human-in-the-loop* (HITL) settings, automated scoring is the first pass and ambiguous cases are escalated to expert annotators.

Lin et al. [10] group chatbot testing methodologies into five families:

* **Usability testing:** Real or alternate users interact with the bot. Evaluators judge clarity, coherence and satisfaction.
* **Simulated-user testing**: Pre-scripted prompts are replayed automatically and outputs are compared with expected answers.
* **AI-planning testing:** Dialogue states are modelled as a planning problem. Test sequences are generated to hit rarely visited branches.
* **Metamorphic testing**: The bot or its input is systematically mutated while expected logical relations hold constant.
* **Paraphrase testing:** Semantically equivalent utterances (spelling errors, syntax changes) probe robustness without altering intent.

# Related Work

## TruthfulQA

TruthfulQA [1] benchmark’s question bank is an 817-item, hand-written corpus that spans 38 everyday themes such as health, finance, law and popular myths. Each item comes with a verified “true” or “false” reference answer plus a citation to a public source like Wikipedia, so the knowledge base is compact yet diverse. Evaluation happens in two flavors. In the open-ended setting every model must generate a free-form answer; in the multiple-choice setting it simply picks between the true and the false reference. Humans play three roles. They wrote every question, supplied the ground-truth answer and later double-rated system outputs for truthfulness and informativeness. Ratings use a continuous 0-to-1 truth score where 0.5 is the pass threshold, allowing researchers to see partial gains before headline “percentage true” figures are derived. Although humans set the gold labels, the authors also trained a GPT-judge that reproduces those labels with up to 96 percent agreement, so large runs can be scored automatically while still tracing back to human ground truth. The strengths here are the citation-backed questions and the fine-grained scale that exposes small improvements, whereas a weakness is that saying “I don’t know” earns the same score as giving a fully correct and informative answer, which may discourage richer model replies.

All tooling lives in a public GitHub repository written in Python. The backend relies on PyTorch and HuggingFace Transformers to load checkpoints, with lightweight helper scripts that call the OpenAI API for GPT-based models and run NumPy-based metrics, so teams only need a standard Python stack to reproduce results. There is no dedicated graphical front-end. Interaction is via command-line scripts or Jupyter notebooks, which keeps the architecture simple but demands basic scripting skills.

Because researchers feed stored questions to a model and compare the output with gold answers, the benchmark clearly follows the Simulated-users testing paradigm: the scripted questions stand in for real users and no AI-planning, metamorphic or paraphrase transformations are applied. Evidence for this appears in the authors’ description of batch-query scripts that loop over the question JSON and write model replies to a CSV before scoring. Strengths of this architecture are its transparency, small codebase and ease of CI integration, while gaps include the lack of a web dashboard and the need for GPU time when evaluating large checkpoints.

## ChatEval

ChatEval [2] keeps its reference data small but varied: it publishes three open-domain prompt sets that cover social-media (Twitter), movie subtitles (OpenSubtitles) and the 200 “Neural Conversational Model” prompts, plus a curated 200-utterance slice of the Dialogue Breakdown Detection Challenge corpus with human reference answers so evaluators compare every system on exactly the same inputs. After a researcher uploads a file of model replies, ChatEval first runs automatic metrics such as lexical diversity, BLEU-2, embedding cosine similarity and perplexity to give a quick quality sketch. It then launches an A/B human study on Amazon Mechanical Turk where three crowd workers see each prompt with two competing responses and pick the better one or declare a tie. These binary votes are aggregated with item-response-theory statistics to rank systems and test significance. Humans therefore design the prompts, supply the gold reference where one exists, and carry out the judgement phase, while automated metrics handle the bulk scoring for large model sets. A strength is this hybrid scoring pipeline that marries cheap automatic measures with consistent human preference data; gaps appear in the low inter-annotator agreement (k between .2 and .54) and the fact that the crowd task measures relative preference rather than absolute factual correctness.

The backend is an open-source Python toolkit that relies on standard libraries: OpenNMT-py for baseline models, NumPy and SciPy for metrics, and a MySQL database served from Google Cloud to host raw ratings and model outputs. Everything is wrapped in lightweight scripts so teams can reproduce results locally or plug them into a CI pipeline. The frontend is a Flask-style web portal with HTML and JavaScript pages for model upload, profile visualization and side-by-side response comparison.

Because evaluation proceeds by replaying fixed prompt files and logging responses, the framework follows the Simulated-users testing methodology. Every user turn is a stored prompt rather than a live participant, and the paper provides scripts that loop over the JSON prompt list to build CSV output files as evidence. It also realizes Usability testing through its controlled A/B human studies that focus on perceived response quality, but it does not employ AI-planning, metamorphic or paraphrase transformations. This architecture is transparent and easy to extend with new datasets, yet its reliance on a web submission workflow means researchers must wait for manual checks before scores are published, and no interactive dashboard exists for drilling into individual worker disagreements.

## FACT-GPT

The authors for FACT-GPT [4] treat the existing global repository of journalist-written fact checks (the ClaimReview database plus Full Fact and AFP articles) as their knowledge base. Each entry is stored as a headline-style claim, a snippet summary, a verdict label such as True, False or Misleading, and a source URL, so the system can look up evidence immediately after a match is found. Evaluation centers on how well the model links a *new* social-media post to the *right* previously-checked claim. The team measures Precision, Recall and F1 for the top-1 suggestion and also reports a “hit-in-top-3” rate that mirrors real newsroom usage. Humans intervene twice. First, two annotators build a gold set of 2,143 Twitter and Reddit posts carefully paired with reference fact checks; Cohen’s k reaches 0.83 which shows high agreement. Second, domain journalists manually judge 100 ambiguous pairs that the model flags as borderline matches. The scoring scale is therefore discrete: exact match, partial match, or no match, but the published leaderboard converts this into standard classification metrics, and the final labels come from human annotators, not the LLM itself. A clear strength is the high-quality human ground truth, while a weakness is reliance on verdict labels that may differ across fact-checking outlets.

The pipeline is implemented in Python. A back-end module calls the OpenAI API to run GPT-3.5 with carefully engineered system and user prompts, then funnels the results into a PyTorch fine-tuning loop that trains a DistilRoBERTa ranker; FAISS is used to index ClaimReview embeddings for fast retrieval. All orchestration, data augmentation and evaluation scripts live in one GitHub repo with less than 1,400 lines of code. There is no bespoke front-end. Fact-checkers interact through a Streamlit dashboard written in 60 lines of Python that lets them paste a claim and view the ranked matches with verdict badges and evidence links.

Because the framework feeds a fixed set of real-world claims to the model and scores its matches against a human gold file, the testing methodology is Simulated-users testing. The authors provide explicit evidence by publishing the JSON script that loops through every tweet in the test set, calls the LLM, and logs matches before computing Precision and Recall. No AI-planning, metamorphic rules or paraphrase generators are employed, which keeps the architecture simple but means the tool cannot yet prove robustness.

## ChatClimate

ChatClimate [5] project’s knowledge base pools only peer-reviewed or highly authoritative sources: six IPCC Assessment Reports, monthly WMO bulletins, 290,000 climate-science abstracts from Web of Science and the thousand most-cited climate papers, with a small adversarial slice from the denial-leaning NIPCC reports for stress tests. Evaluation uses 170 statements that had already been annotated by Climate Feedback scientists as “accurate”, “misleading”, or “incorrect”. Those expert judgements form the gold truth. Three GPT-4 “Advocate” agents, each restricted to one corpus, debate a claim, after which a GPT-4 “Mediator” blends their verdicts into a four-level credibility label that maps down to a simple credible / not-credible split for headline scores. Humans are involved only in building the gold set and seeding the corpora. Once those are fixed, the LLM agents assign every score automatically. On the common two-class scale the system reaches 96 % accuracy, while on the five-class tier it achieves 73 %, sixteen points above a single-LLM baseline. Strengths are the citation-backed evidence each Advocate returns and the debate step that surfaces conflicting viewpoints. Gaps lie in dependency on a closed-source GPT-4 and the risk that static corpora lag behind fast-moving climate research.

All code is written in Python and will be released on GitHub. Each Advocate pairs a retrieval module, ElasticSearch indices with BM25 ranking, against the OpenAI GPT-4 API for reasoning; the Mediator is another GPT-4 instance that issues follow-up prompts until the Advocates converge. Results are returned as JSON and visualized in Jupyter notebooks, so there is no standalone web front-end. Users need only the Python stack plus an OpenAI key.

Because the evaluation simply feeds a fixed list of expert-written claims to the pipeline and compares its labels with the human gold file, the work employs Simulated-users testing. The paper’s Appendix lists the batch-processing script that iterates over the 170 statements and logs accuracy after each run, showing no evidence of AI-planning search trees, metamorphic input mutations or paraphrase variants. The backend’s modular RAG design makes reproduction straightforward and lets researchers swap corpora easily. However, there are still some downsides. Running large Advocate models can be expensive because they need GPUs, and there’s no easy-to-use dashboard to explore errors.

## MutaBot

In MutaBot [3], all intents, entities and example utterances are parsed and stored so they can be altered by the mutation engine. Evaluation follows classic mutation-testing logic. MutaBot generates hundreds of “mutants” by applying 16 operators such as swapping required entities, deleting training phrases or renaming intents; it then reruns the developer’s existing Botium test-suite against every mutant and records whether the tests “kill” it. Humans step in only twice, first to design the reference test cases and later to inspect mutants that survive all tests and decide whether they are *equivalent* (functionally identical to the original) or reveal a real gap. The scoring scale is a single mutation score between 0 and 1 calculated as killed / total mutants, so a suite that kills 190 of 200 mutants scores 0.95. Those scores are computed automatically by the framework, but the final decision on equivalent mutants is left to the tester, which is a strength for precision yet a weakness for scalability when thousands of mutants are generated.

MutaBot’s backend is a set of Python 3 modules that call the Dialogflow REST API to clone agents and a Node.js wrapper that drives Botium CLI for test execution. Results are aggregated with pandas, a Python library, and exported as JSON and CSV so they slot easily into CI pipelines. There is no heavyweight front-end. Developers trigger runs from a command-line script and open the generated HTML coverage report that is rendered with the JavaScript library Chart.js, keeping the stack lightweight but offering limited interactive drill-down.

Because the tool mutates the bot specification and then replays scripted conversations, it clearly follows the Metamorphic testing methodology and counts as Simulated-users testing. The paper evidences this with an experiment on three real Dialogflow agents where 1,240 mutants were created and the Botium test-suites automatically exercised every path. A notable strength is that the approach surfaces silent intent-mapping errors that traditional accuracy metrics miss, while gaps include the manual burden of filtering equivalent mutants and the need for developers to maintain a robust Botium test-suite before mutation testing can pay off.

## BoTest

In BoTest [6], the authors begin with a knowledge base consisting of 400 “clean” small-talk utterances that their Azure LUIS bot (*ChitChatBot*) already answers correctly; each utterance is labelled with its intended intent and any required entities. From this seed set they hand-craft two divergent input classes, a syntactic variant that alters function-words (e.g., “*Who’s on telly tonight?*” 🡪 “*Who is on TV tonight?*”) and a colloquial variant that swaps in Irish slang (e.g., “*Tell me the weather*” 🡪 “*What’s the craic with the weather?*”). During evaluation every divergent input is fed to the bot and the returned intent is compared with the original clean label, producing a simple pass / fail tally such as “26 of 48 variants handled”. Humans are involved at two points: they write the baseline utterances and their divergent counterparts, then manually review the log to decide whether borderline answers still satisfy the intent. Because the metric is binary, the final truth score is a proportion (0 to 1) of variants correctly understood. Those scores are calculated automatically by the BoTest script, but the ground-truth decision on each utterance is ultimately made by the human testers. Strengths here are the transparent, easy-to-interpret metric and the focus on real wording tweaks that often break chatbots, while the main gap is the manual effort needed to craft divergents for every new domain or language .

BoTest’s backend is a set of Python 3 modules that export three functions: *generate\_variants*, *run\_dialogue*, and *evaluate*, and a small command-line driver. The script calls the Azure LUIS REST API to send each utterance, stores JSON responses in SQLite via SQLAlchemy, and aggregates results with pandas. No GPUs or external libraries beyond *requests* and *pandas* are required . There is no dedicated front-end as testers open the generated CSV or an optional Jupyter notebook that plots variant-type accuracy using Matplotlib.

Because the framework replays a scripted set of user messages and relies on meaning-preserving rewrites to stress the bot, its primary testing methodology is Simulated-users testing combined with Paraphrase testing as the paper explicitly lists “syntactic variant” and “colloquial variant” generators as the core evidence of this focus. BoTest’s architecture is therefore lightweight and CI-friendly, yet the absence of a richer web dashboard limits quick error drilling and the lack of automatic variant generation means coverage depends on the creativity of the human author.

## DialTest

DialTest [7] starts from the *developer’s own dialogue data*, the clean user utterances, intents and slot values that were already used to train or validate the RNN policy. That seed set acts as the knowledge base against which all tests are judged. The tool then applies ten transformation operators, word-insertion, deletion, synonym replacement, spelling corruption, polite-phrase padding and more, to create thousands of variants for each utterance. Every variant is replayed through the live bot, and the predicted intent/slot combination is compared with the ground-truth label of its clean parent. A run therefore yields a simple binary verdict for each test: the variant is either still understood or it is not. Humans appear only at two points: first when they prepare the clean, labelled seed utterances and later when they inspect “surviving” failures to decide whether a mis-routed response is genuinely wrong or an acceptable alternative. Scores are reported as a single mutation-style number between 0 and 1, the fraction of generated variants that the bot handled correctly, so assessment is largely automated yet still traceable to human labels where edge cases arise. The fine-grained mutation score highlights incremental robustness gains, but coverage is limited by the quality and scope of the original seed utterances.

The implementation is split across Python 3 modules. Back-end code loads the RNN model in TensorFlow/Keras, drives test execution with the DialTest Engine written in pure Python, and stores results in pandas data frames. Optional FAISS indexing speeds up synonym look-ups. No dedicated front-end is shipped. Developers run a command-line script that writes CSV and an HTML coverage report rendered with Chart.js, keeping the stack light but offering only static drill-down.

Because DialTest first *mutates* existing utterances and then replays them automatically, its methodology combines Simulated-users testing (the scripted replay) with Metamorphic testing (each operator expresses a metamorphic relation such as “adding a polite prefix should not change intent”). The paper evidences this by demonstrating how word-insertion and synonym-replacement operators expose hidden intent-classification bugs that ordinary accuracy metrics miss. Strengths of the architecture are its Python core, seamless CI integration and ability to pinpoint brittle intent definitions, while notable gaps include the manual effort of filtering equivalent mutants and the absence of a richer web dashboard for live error exploration.

## Bottester

In Bottester [8], the authors treat each scenario file, a CSV list of question 🡨🡪 reference-answer pairs, as the knowledge base. In their showcase they imported 491 Portuguese finance questions and gold answers from the *CognIA* advisor so that every replayed query already has an authoritative reply to compare against. During a run Bottester replays every question through the bot’s public API, captures the textual response and evaluates it with a binary oracle: an answer passes only when it matches the reference string exactly. At the same time the tool logs end-to-end latency, answer length and repetition counts so testers can notice verbose or slow replies. Humans are involved up front to prepare scenario files and gold answers; after execution they may inspect any failures the oracle flags, but all scoring is automatic. The truth scale is therefore simple pass/fail and the overall score becomes a proportion of correct replies (e.g., 472 / 491 ≈ 0.96) computed entirely by the framework. Strengths are the realistic, interface-level replay and the extra UX metrics. Gaps include the coarse exact-match rule that marks partially correct paraphrases as wrong.

Bottester is organized in three layers: a file-based scenario loader, an execution engine that fires REST calls to the target chatbot, and a desktop dashboard that visualizes per-question and per-agent metrics. The paper does not specify the exact programming languages for the back-end nor front-end.

Because testers replay stored conversations rather than live users, the framework clearly follows Simulated-users testing. It also injects variant inputs such as typos and synonym replacements that preserve meaning but try to break the bot, so it embodies both Metamorphic testing and Paraphrase testing. The evidence is an experiment where typo-laden questions exposed hidden intent failures that the clean set missed. The thin codebase, CSV workflow and exportable JSON results make it easy to slot into automated pipelines; weaknesses are the lack of a web dashboard for interactive drill-down and reliance on verbatim matching, which overlooks semantically equivalent answers.

## OggyBug

OggyBug [9] treats the test-author’s own conversation scripts as its knowledge base. Each script is a JSON array holding a user message, the bot’s expected reply and optional context such as slot values, so every test already carries a ground-truth answer. During a run OggyBug replays each script through the live bot’s public API, records the response and checks it with an oracle that supports three verb levels to judge success: exact, contains and regex. Humans therefore intervene only twice. They write or refine the scripts and, after a run, inspect any mismatches the oracle flags. The tool logs a binary pass/fail for every turn, then rolls these into a scenario-score between 0 and 1 (e.g., 47 / 50 = 0.94). Those scores are generated automatically, but final sign-off on borderline mismatches or acceptable paraphrases rests with the tester.

The back-end is written in Python 3.8. A core test runner built with *requests*, *asyncio* and *pandas* fires API calls, timestamps responses and streams raw logs to a SQLite database. A small Node.js/Express micro-service listens for webhook events so the same runner can be triggered from CI systems such as GitHub Actions. On the front-end a Vue.js single-page dashboard renders scenario heat-maps and per-turn diffs via the Chart.js and Vue-Table libraries; testers can click any failed turn to view the raw JSON exchange.

Because OggyBug feeds stored user messages to the bot, its primary methodology is Simulated-users testing. By letting testers embed regex or synonym patterns that should still pass, it also counts as Metamorphic testing. The paper evidences this with a case study on an IBM Watson assistant where a single regex pattern (“hi|hello|hey”) killed eight hidden intent bugs that literal tests missed. Strengths of the architecture are its language-agnostic JSON scripts, CI-ready webhooks and visual viewer. Weaknesses include reliance on manual script upkeep and the absence of automated paraphrase generation, which limits robustness coverage.

# Discussion

Refer to Table 1 which shows a comparative analysis of existing frameworks. Taken together, the nine systems trace a continuum from pure truth-stress benchmarks (TruthfulQA, ChatClimate) through evaluation hubs for many bots (ChatEval, FACT-GPT) to robustness and regression-testing toolkits (MutaBot, BoTest, DialTest, Bottester, OggyBug). A first pattern that emerges is the **knowledge-base scope**. TruthfulQA and ChatClimate ship carefully authored corpora with cited references, so every run is grounded in stable evidence . By contrast, FACT-GPT and ChatEval depend on user-supplied logs. They gain flexibility but forfeit cross-run comparability unless users share the same prompt sets.

A second axis is **human effort**. TruthfulQA, ChatClimate and ChatEval invest heavily in expert or crowd annotation, which secures trustworthy gold labels but also caps dataset size and update speed. FACT-GPT narrows this bottleneck by letting an LLM assign provisional matches that humans audit in batches, an approach echoed by Bottester’s option to review only surviving failures. The three tools (MutaBot, BoTest, DialTest) work differently. After setting up the rules, they can generate many test cases on their own. Humans only need to step in to check any unclear results. For teams with few experts, these tools save the most time, though they still need a few good example phrases to begin with.

From a **scoring-scale perspective**, truth benchmarks adopt *graded* scales (0–1 or four-level credibility), enabling nuanced tracking of partial gains, whereas all five robustness frameworks default to *binary pass-fail*. This makes it easy to use with CI pipelines, but it might hide small improvements like answers that are almost correct but still not right. In DialTest’s extra *coverage* metrics, scoring remains numeric yet capture incremental progress.

On the backend, all tools use **Python**, showing how central it is for NLP testing. But the frontends differ: some tools (like ChatEval and OggyBug) use web interfaces, making them easier for non-coders. Others (like TruthfulQA, MutaBot, and BoTest) only work through the command line, which fits well with DevOps but needs coding skills. So far, no tool combines both styles, making it harder for researchers and teams to work together smoothly. Also, most tools don’t show results visually in real time, only Bottester has a dashboard that tracks delays. This shows a clear need for better interfaces that can highlight common errors and tricky test areas.

Mapping against the **five testing-method taxonomy** shows healthy coverage of *Simulated-user* and *Metamorphic/Paraphrase* methods but an absence of formal *AI-planning* or large-scale *Usability* experiments beyond ChatEval’s crowd Likert ratings. This skew implies that current practice focuses on functional correctness and robustness, leaving conversational flow optimization and goal-oriented planning comparatively under-explored. Bridging these blind spots, for example adding plan-based scenario generators to MutaBot or integrating real-time UX analytics into OggyBug, would round out the toolkit landscape and move the field closer to a holistic quality assurance for conversational AI.

1. Comparative Analysis of Existing Frameworks

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tool / Paper** | **Knowledge base** | **Evaluation method** | **Human involvement** | **Truth-score scale** | **Who assigns scores?** | **Backend**  **(languages / stack)** | **Front-end**  **(languages / stack)** | **Testing methodology** |
| TruthfulQA [1] | 817 hand-written Q-A pairs across 38 topics, each with a cited source | Open-ended generation and MC-1 multiple-choice; answers compared with gold or ranked by GPT judge | Write questions, craft gold answers, rate system outputs | 0-1 continuous (≥ 0.5 = true) | Initial labels and spot-checks by humans, bulk scoring by GPT-judge | Python + PyTorch + Hugging-Face scripts, JSON data | No dedicated UI - CLI / Jupyter notebooks | Simulated-users |
| ChatEval [2] | User-uploaded dialogue turns and reference replies | Automatic metrics (BLEU, METEOR) plus 1-5 Likert crowd ratings | Crowd workers rate coherence, relevance etc. | 1-5 Likert per dimension | Humans (crowd) + automatic scripts | Python (Flask, Celery) + PostgreSQL | Web portal with HTML/CSS/JS (React) | Usability,  Simulated-users |
| FACT-GPT [4] | Claim bank mined from fact-checking sites and social media | LLM finds nearest fact-checked claim, returns stance + evidence | Experts curate seed claims and validate matches | Binary match / stance confidence | LLM assigns, human validators sample-check | Python (OpenAI + Hugging-Face) | Notebook / REST demo, no full GUI | Simulated-users |
| ChatClimate [5] | IPCC AR6, WMO bulletins, peer-reviewed abstracts | Retrieval-augmented GPT-4 answers; climate scientists grade credibility | Scientists design rubric and score 170 test statements | Four-level rubric 🡪 binary credible / not | Humans for official scores, LLM mediator runs automatically | Python (LangChain, OpenAI) + Elasticsearch | Slack-bot & Jupyter prototypes | Simulated-users |
| MutaBot [3] | Dialogflow bot intents; mutants of training set and code | Applies 18 mutation operators, reruns test suite, computes mutation score | Devs inspect surviving mutants if needed | Killed / survived 🡪 % mutation score | Automated harness; humans analyze hard cases | Java (Spoon) + Python drivers | CLI reports (no GUI) | Metamorphic,  Simulated-users |
| BoTest [6] | 400 “working” utterances for a small-talk bot | Generates divergent (paraphrase / typo) inputs, checks intent match | Authors craft divergents and tally results | Pass / fail counts | Humans score but script flags mismatches | Python scripts | None (CSV output) | Paraphrase,  Simulated-users |
| DialTest [7] | RNN dialogue model’s seed utterances | Ten transformation operators create thousands of tests. Coverage and failure rate reported | Humans design operators & inspect failures | Pass / fail + intent/slot coverage % | Automated oracle; manual review optional | Python (PyTorch) | CSV/HTML reports | Metamorphic, Simulated-users |
| Bottester [8] | CSV scenario files with expected replies | Simulated chat, logs correctness, latency, repetitions | Authors prepare scenarios; optional manual review | Binary pass / fail per turn | Automated; humans create scenarios | Python executor + SQLite | Desktop dashboard | Simulated-users,  Metamorphic,  Paraphrase |
| OggyBug [9] | JSON test scenarios for IBM Watson / Rasa | REST runner replays flows, checks intents, entities, context | QA staff write scenarios and set assertions | Pass / fail per step; coverage summary | Automated engine; humans write cases | Python +Django REST, Selenium hooks | Web GUI (Bootstrap, JS) | Simulated-users, Metamorphic |
| FactBridgeAI | Versioned test cases + expected replies stored in PostgreSQL | Docker runner feeds cases to bot container; LLM verifier compares answers; results logged | Experts compose tests; intervene only when verifier low confidence or disagreement arises | 0-1 confidence plus configurable pass/fail threshold | LLM verifier primary; human annotators validate flagged items | Docker-Compose micro-services (TBD) | Web GUI (TBD) | Simulated-users by default; supports Metamorphic & Paraphrase through LLM verifier; Usability via web analytics |

# Towards a Semi-Automated Human-in-the-Loop Fact-Checking Framework

Existing frameworks rarely hand off “only” the truly doubtful cases to humans. ChatClimate lets climate scientists grade GPT-4 answers on a four-level credibility scale, and TruthfulQA shows that a GPT-judge can replicate human labels with 96 percent agreement, yet both still rely on humans to label every answer at least once, then use the machine only for bulk rescoring. No tool in the set maintains a live confidence score per interaction and routes just the low-confidence slices to expert reviewers, so human effort remains front-loaded instead of being spent where it matters most.

Long-term traceability is also weak. Bottester logs scenario files and run data in a local SQLite database, and OggyBug keeps JSON test flows plus historic pass/fail counts, but these stores live inside project folders and are overwritten or forked whenever a new model version is tried. Other systems dump CSV files or dashboards. Because none offers a shared, versioned repository that records prompts, model outputs, scores and subsequent human corrections in one place, researchers cannot track drift across iterations or easily compare competing agents on a single timeline.

Reproducibility across environments is the third missing piece. ChatEval ships its evaluation server in a Docker image and MutaBot includes examples of GitLab-CI scripts, yet the chatbot under test is left to run on whatever host libraries happen to be installed. The remaining frameworks either require manual Python environments (TruthfulQA, BoTest, DialTest) or assume a vendor platform such as Dialogflow. Without a prescribed container orchestration that snapshots both the agent and the scorer, rerunning last quarter’s benchmark after a dependency upgrade becomes a challenge.

None of the reviewed solutions delivers the full trio of (i) confidence-based human escalation, (ii) a durable, queryable results repository, and (iii) container-level reproducibility. Each tackles a slice of scoring accuracy, robustness, or usability, but leaves gaps that make life-cycle fact-checking cumbersome. These blind spots motivate the design of FactBridgeAI, whose central goal is to weave the three missing capabilities into one coherent, traceable and easily redeployable testing loop. Refer to the bottom of Table 1 to see how FactBridgeAI compares with the existing tools.

Our solution, FactBridgeAI, fills these gaps using a Docker-based micro-service stack. It uses a central database to store test cases, expected answers, chatbot versions, and test results, solving the problem of missing records found in tools like ChatEval and FACT-GPT. A runner service spins up the chatbot under test inside its own container, feeds the stored prompts, and logs raw outputs back to the database, mirroring the simulated-user strategy proven effective by TruthfulQA and Bottester. Another service scores the replies using a large language model (LLM), much like TruthfulQA’s GPT-judge or FACT-GPT’s fact-checking step. These scores are saved with the conversation so they can be reviewed later. If the model isn’t confident about a score, the system flags it for a human to check using a simple web interface, so experts only review cases the machine is unsure about. Since everything is run using Docker Compose, researchers can exactly replay old tests, avoiding the setup issues seen in MutaBot and DialTest.

# Future Work

FactBridgeAI is intentionally modular so that researchers can swap components as new techniques mature, yet several research issues remain open:

* **Verifier transparency and bias**: Today’s LLM‐based scorers still act as black boxes. Integrating explanation modules or rule-augmented chains will help annotators trust confidence scores before escalation.
* **Dynamic threshold tuning**: By learning from past disagreement patterns between the LLM and experts, the system could raise or lower its confidence cut-offs to keep human workload stable under changing model quality.
* **Multilingual and multimodal coverage**: Extending the database schema to accommodate non-English prompts, images or tables, and plugging in OCR or vision, language verifiers, would make the framework relevant beyond text-only English chatbots.
* **Automated scenario generation**: Planning-based or data-mined paraphrase synthesis could enrich the test repository continuously, closing the gap that DialTest and MutaBot only partly address.
* **CI/CD integration**: DevOps integration with cloud container registries and security-scoped execution (e.g., Kubernetes pods with network guards) will be essential before enterprises adopt the loop in production.

# Conclusion

This review mapped nine peer-reviewed chatbot-testing frameworks across eight technical dimensions and five testing methodologies. TruthfulQA and ChatClimate excel at curated, citation-linked corpora; ChatEval pioneers scalable human ratings; FACT-GPT offers LLM-powered claim matching; and MutaBot, BoTest, DialTest, Bottester and OggyBug expose robustness weaknesses through scripted or mutated inputs. Yet taken together they still leave three critical gaps: no confidence-driven escalation workflow, no long-term, versioned repository of test artifacts and outcomes, and no containerized mechanism that guarantees exact re-execution across model iterations. FactBridgeAI is proposed to bridge these gaps. It stores every prompt, response and score in a relational database, executes both the chatbot and the verifier inside Docker containers for reproducibility, and escalates only uncertain cases to human annotators, thereby combining the speed of automated scoring with the depth, auditability and lifecycle traceability of expert review. FactBridgeAI should serve as a baseline fostering an fact-checking for AI chatbots.

##### References

[1] S. Lin, J. Hilton, and O. Evans, “TruthfulQA: Measuring How Models Mimic Human Falsehoods,” in Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), S. Muresan, P. Nakov, and A. Villavicencio, Eds., Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 3214–3252. doi: 10.18653/v1/2022.acl-long.229.

[2] J. Sedoc, D. Ippolito, A. Kirubarajan, J. Thirani, L. Ungar, and C. Callison-Burch, “ChatEval: A Tool for Chatbot Evaluation,” in Proceedings of the 2019 Conference of the North, Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 60–65. doi: 10.18653/v1/N19-4011.

[3] M. F. Urrico, D. Clerissi, and L. Mariani, “MutaBot: A Mutation Testing Approach for Chatbots,” in Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, in ICSE-Companion ’24. New York, NY, USA: Association for Computing Machinery, May 2024, pp. 79–83. doi: 10.1145/3639478.3640032.

[4] E. C. Choi and E. Ferrara, “FACT-GPT: Fact-Checking Augmentation via Claim Matching with LLMs,” in Companion Proceedings of the ACM Web Conference 2024, Singapore Singapore: ACM, May 2024, pp. 883–886. doi: 10.1145/3589335.3651504.

[5] S. A. Vaghefi et al., “ChatClimate: Grounding conversational AI in climate science,” Commun Earth Environ, vol. 4, no. 1, p. 480, Dec. 2023, doi: 10.1038/s43247-023-01084-x.

[6] E. Ruane, T. Faure, R. Smith, D. Bean, J. Carson-Berndsen, and A. Ventresque, “BoTest: a Framework to Test the Quality of Conversational Agents Using Divergent Input Examples,” in Companion Proceedings of the 23rd International Conference on Intelligent User Interfaces, in IUI ’18 Companion. New York, NY, USA: Association for Computing Machinery, Mar. 2018, pp. 1–2. doi: 10.1145/3180308.3180373.

[7] Z. Liu, Y. Feng, and Z. Chen, “DialTest: automated testing for recurrent-neural-network-driven dialogue systems,” in Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Denmark: ACM, Jul. 2021, pp. 115–126. doi: 10.1145/3460319.3464829.

[8] M. Vasconcelos, H. Candello, C. Pinhanez, and T. dos Santos, “Bottester: Testing Conversational Systems with Simulated Users,” in Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems, in IHC ’17. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 1–4. doi: 10.1145/3160504.3160584.

[9] M. B. D. Santos, A. P. C. C. Furtado, S. C. Nogueira, and D. D. Moreira, “OggyBug: A Test Automation Tool in Chatbots,” in Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing, Natal Brazil: ACM, Oct. 2020, pp. 79–87. doi: 10.1145/3425174.3425230.

[10] X. Li, C. Tao, J. Gao, and H. Guo, “A Review of Quality Assurance Research of Dialogue Systems,” in 2022 IEEE International Conference On Artificial Intelligence Testing (AITest), Aug. 2022, pp. 87–94. doi: 10.1109/AITest55621.2022.00021.