Proof Repair Utilizing Large Language Models: A Case Study on the Copland Remote Attestation Proofbase

Amer Tahat[1], David Hardin[1], Adam Petz[2], and Perry Alexander[2]

[1] Collins Aerospace
Cedar Rapids, IA 52498 USA
{amer.tahat,david.hardin}@collins.com
[2] Institute for Information Sciences
The University of Kansas
Lawrence, KS 66045 USA
{ampetz,palexand}@ku.edu

Abstract. Large Language Model (LLM) Artificial Intelligence (AI) systems have generated significant enthusiasm in the computer science research community for their potential in various computer language processing tasks, such as source code generation and source-to-source translation. We are particularly interested in using LLMs for automated theorem proving, specifically for proof repair. To this end, we introduce CoqDog Copilot, which leverages the neuro-symbolic interplay between generative AI and the Coq theorem prover to form a productive "generate-and-test" loop, incrementally improving proofs based on failure information and human hints until valid proofs are achieved. Our research introduces innovative solutions to critical challenges in developing CoqDog Copilot, including addressing context limitations, enhancing the soundness of recommendation systems, defining effective metrics for measuring proof repair progress, and designing a statistically robust evaluation system for conversational quality assessment. We present a comprehensive evaluation of CoqDog Copilot's performance in proof repair across multiple samples from the Copland Coq proofbase, which consists of a total of 21,000 lines of Coq code. We have attained in excess of 60% accuracy for proof generation using GPT-4 in one 'shot', with approximately 30% more lemmas proved given one additional user prompt (yielding 90% correctness overall). With three 'shots', the overall proof correctness rate increases to 97%. We could generate Coq proofs with up to 50 proof steps using this technique. Our LLM-generated proofbase currently consists of over 1,400 lines of Copland Coq source.

1 Introduction

Large Language Model (LLM) Artificial Intelligence (AI) systems have ignited great excitement in the computer science research community for their potential to perform a number of computer language processing tasks, including source code generation from natural language descriptions, source-to-source translation, and the like. LLMs do have a significant downside, however: they are apt to produce "hallucinations", textual output that appears convincing at first glance, but is in fact, nonsensical. This is especially problematic when it comes to software synthesis for high-assurance applications. Recent studies on the use of GitHub Copilot [15], an LLM-based coding assistant, show that 40% of the generated code contained serious vulnerabilities. In a recent comparative study, Perry *et al.* report that developers who made use of OpenAI's Codex LLM-based coding assistant on security-related coding assignments "wrote significantly less secure code than those without access." [18]

We are interested in the use of LLMs for automated theorem proving, particularly proof repair. Proof repair is a great use case for LLMs, because the neuro-symbolic interplay between the generative AI and the automated theorem prover form a productive "generate-and-test" loop, with the LLM incrementally modifying its proof attempts based on proof failure information from the theorem prover, as well as hints from the human user, until a valid proof is achieved. The automated theorem prover thus acts a foolproof "hallucination detector"; the proof will fail if any of the LLM-generated steps are nonsense. We have focused on a particular use case: the Copland Semantic Remote Attestation proofbase developed by researchers at the University of Kansas. This paper summarizes the results of our use of LLMs for proof repair for the Copland proofbase. In pursuit of this goal, in this paper we introduce CoqDog Copilot, (see Figure 1 and Figure 2). We summarize our contributions as follows:

1.1 Contributions of this Paper

1. Our research addresses critical challenges in developing a generative AI assistant for proof synthesis and repair, including:
   - Introducing CoqDog Copilot, a web application for proof synthesis and repair using the Coq theorem prover (see Figure 1).
   - Proposing an innovative approach to address the copilot's context limitation (Figure 3).
   - Developing a sound recommendation system supported by a ranking mechanism to prevent logically unsound recommendations.
   - Establishing effective metrics for measuring proof repair progress.
   - Designing a statistically robust evaluation system for assessing conversational quality (Figure 5).
2. We present a comprehensive evaluation of CoqDog Copilot's performance in proof repair across multiple samples, as shown in Figure 4 from the Copland Coq proofbase, which consists of a total of 21,000 lines of Coq code.

1.2 Paper Organization

In Sections 1 to 4, we provide a brief introduction, outline our motivations, and offer background information on Copland attestation protocols and their development challenges. In Section 5, we present our proof repair system using LLMs. Section 6 explores the contextual prompt constraints problem and our approach to overcome it. Section 7 discusses our Conversation Quality Assessment System (CQAS) for evaluating proof repair quality. Finally, in Section 8, we summarize our experimental evaluations, key results, highlight current limitations, and suggest future work. Related work and main conclusions are discussed in Section 9 and Section 10, respectively. Finally, Appendix A provides statistical details of our CQAS system's experimental evaluation on the Copland proofbase.

Remark 1. We implemented our CoqDog copilot as a web application, under open source terms. Contact the second author for more details.

2 Copland: Semantic Remote Attestation

Copland [20,21] is a domain-specific language designed for describing, analyzing and executing attestation protocols. Its formal semantics, specified using the Coq theorem prover, defines evaluation, sequencing, and dispatch of measurements resulting in evidence describing a system's state. That evidence is in turn appraised to determine if and how an external system will interact with it. Copland implementation code is automatically generated from the Coq specification via verified synthesis techniques, providing a high-assurance link between specification and realization. Copland protocols may be provisioned early in the development of the system, evolve as the system develops, and continue to function once the system is deployed, re-developed, re-deployed, *etc.* Thus the Copland protocol proofs for these "Lifecycle Attestation" use cases need to evolve as the system design changes. This has led us to perform experiments to determine whether LLM technology could be used to perform automated proof repair for evolving formal attestation protocol specifications.

3 High-Assurance Model-Based System Development

Successful large model-based system developments invariably take the form of *composable*, *accurate* sub-models that developers can readily combine according to well-defined assume/guarantee contracts (for a practical example of this approach, see [4]). By *accurate*, we mean that a model accurately reflects the as-built system and that the model is actively maintained during initial system development, as well as subsequent modification; a model that maintains accuracy in the face of modification is called *resilient.* Engineers will continue to develop and maintain such a model if it is *synthesizable*; that is, it automatically produces (at least) a skeletal implementation that can be integrated with detailed functional components as they are developed (see [2] for an example of this type of high-level system integration tool). This automation of system integration avoids time-consuming errors associated with manual "big bang" system integration efforts under time pressure. Synthesizable models also allow for automated generation of simulations that execute on a host-based operating system or virtual machine.

Engineers are motivated to develop and maintain a model that is *observable/analyzable*, *i.e.*, one that provides useful, accurate evidence of whether the developing model satisfies its key requirements-derived properties, both at development time and at run time. Additionally, it should support compositional analyses to support activities such as schedulability analysis, information flow analysis, cyber vulnerability analysis, as well as detailed change impact analysis when design modifications are contemplated.

Finally, an *accreditable/certifiable* model is one that directly supports the accreditation/certification process through formal analysis, direct integration with assurance cases and evidence generation, such as the Resolute assurance case tool reported on in [1].

4 Lifecycle Remote Attestation

As a model of transforming system observation we propose extending the concepts of *semantic remote attestation* to the system design process; in effect, providing "birth to death" attestation. Recall that semantic remote attestation is a process of gathering evidence in support of a trust decision. *Measurements* gather evidence, *attestation* bundles evidence with meta-evidence, and *appraisal* assesses evidence to determine trust.

Typically, a remote attestation system is a one-off system focused on an implemented, operational system. Measurements make observations during system boot and execution. They do not extend to system design, provenance, certification or postmortem and are rarely associated with specific system requirements. Furthermore, attestation systems do not evolve with their associated systems.

Our vision is to expand the concepts of measurement and attestation, making attestation first-class in the system-level design process. Attestation will apply from initial system specification and evolve with the design through certification, deployment, and system update cycles. As the system evolves, so will our mechanisms for observation while maintaining a semantic connection to system requirements. Specifically, we will: (i) define static measurement; (ii) integrate static measurement into attestation semantics and interpretation; (iii) transform attestation using static measurement into attestation using dynamic measurement; while (iv) providing trustworthy synthesis of dynamic remote attestation and measurement components using formal methods techniques.

5 Lifecycle Attestation Proof Repair using Large Language Models

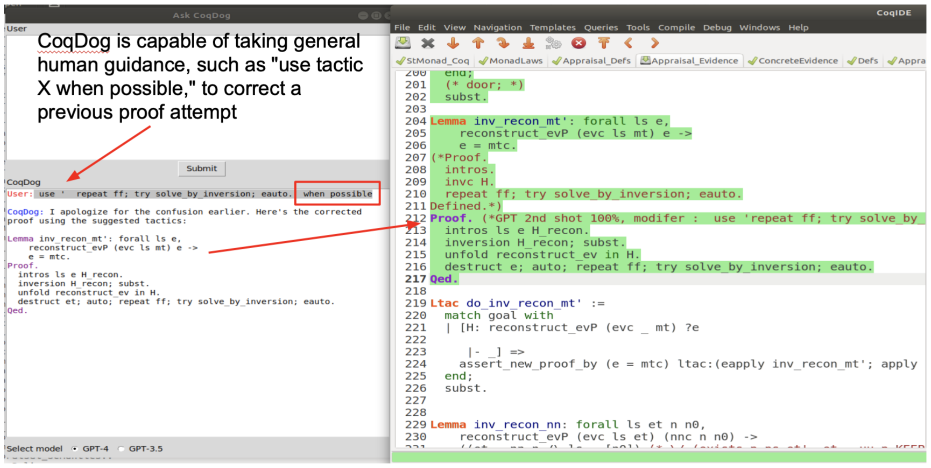
Because Lifecycle Attestation applies to a system under development, the attendant measurement and attestation protocols, specified in Coq (from which the source code is automatically generated), will likely need to evolve as the system design changes. Further, since proof development for Copland protocols requires significant time and expertise, we needed to find a way to reduce the time required to re-prove Copland in a slightly changed environment, and to allow non-Copland experts to oversee the proofs, with machine assistance. This problem, called "proof repair", is becoming increasingly important as proofbases grow and evolve.

We hypothesized that we could utilize Large Language Model (LLM) Artificial Intelligence technology to develop an "AI Assistant" for Coq, which we have named CoqDog (the 'Dog' metaphor suggests a helpful, energetic entity that aims to please). CoqDog is thus similar to many LLM-based "co-pilots" in areas such as software development.

We quickly noted that we could evaluate CoqDog by feeding it lemma conjectures for Copland, but without the proof steps that Coq proof developers compose to prove that those conjectures are mathematically correct, and ask the LLM AI to come up with its own sequences of proof steps. We could then submit these candidate proofs to the Coq theorem prover, and see whether the AI-generated proof steps actually resulted in a proof. This, as it turns out, is a wonderful use case for LLMs, because LLMs have developed a (well-deserved) reputation for producing "hallucinations", textual output that appears convincing at first glance, but is in fact, nonsensical. (NB: This is a real concern, for example, when attempting to use LLMs to compose software, because the generated code may well compile, yet still be wildly incorrect, with no way to tell, absent detailed testing and line-by-line code inspections, whether the generated code 'works', whether it may have introduced a malicious backdoor, etc.) By submitting the LLMs candidate proofs to the Coq theorem prover, we are able to invoke a foolproof "hallucination detector", as the proof will fail if any of the AI-generated steps are nonsense.

The Collins team that developed CoqDog and performed the evaluation on the Copland proofbase were experienced users of interactive theorem provers, but had little experience with LLMs, the Coq theorem prover, or the details of Copland. At first, the team thought that we would need to develop a significant training set in order to yield sensible results from the LLM. But we were pleasantly surprised, especially using OpenAI GPT-4 [12]. GPT-4 began to produce valid proofs on small examples almost immediately, with no training. Surprisingly, it recognized Coq input specifications, and we eventually stopped prompting it to produce Coq proofs. Before long, as the team learned the basics of prompting the system, GPT-4 was producing a high percentage of correct Coq proof sequences, and succeeding on proofs as long as 50 steps.

It turned out the Collins team's lack of experience with Coq or Copland was a benefit for our experiments, as it kept our prompting approach simple, as well as representative of what a new CoqDog user would attempt. One obvious second "shot" was to prompt the chatbot with error messages from Coq when attempting to prove the initial GPT-produced proof, to which the chatbot responded with an improved proof in many instances. The chatbot also reacted well to additional prompts to use particular Coq primitives in its proofs (see Figure 1), thereby improving the quality of its proof output. It also, based on the structure of the Copland proofs presented as input, began to imitate the style of the KU


Ask CoqDog
User
CoqDog is capable of taking general human guidance, such as "use tactic X when possible," to correct a previous proof attempt
Submit
CoqDog
Select model
GPT-4
GPT-3.5
CoqIDE
File Edit View Navigation Templates Queries Tools Compile Debug Windows Help
StMonad_Coq
MonadLaws
Appraisal_Defs
Appraisal_Evidence
ConcreteEvidence
Defs
Appra

```
User: use '  repeat ff; try solve_by_inversion; eauto.  when possible

CoqDog: I apologize for the confusion earlier. Here's the corrected
proof using the suggested tactics:

Lemma inv_recon_mt': forall ls e,
    reconstruct_evP (evc ls mt) e ->
    e = mtc.
Proof.
  intros ls e H_recon.
  inversion H_recon; subst.
  unfold reconstruct_ev in H.
  destruct et; auto; repeat ff; try solve_by_inversion; eauto.
Qed.
```

```
  (* door; *)
  subst.

Lemma inv_recon_mt': forall ls e,
    reconstruct_evP (evc ls mt) e ->
    e = mtc.
(*Proof.
  intros.
  invc H.
  repeat ff; try solve_by_inversion; eauto.
Defined.*)
Proof. (*GPT 2nd shot 100%, modifer :  use 'repeat ff; try solve_by_
  intros ls e H_recon.
  inversion H_recon; subst.
  unfold reconstruct_ev in H.
  destruct e; auto; repeat ff; try solve_by_inversion; eauto.
Qed.

Ltac do_inv_recon_mt' :=
  match goal with
  | [H: reconstruct_evP (evc _ mt) ?e

     |- _] =>
    assert_new_proof_by (e = mtc) ltac:(eapply inv_recon_mt'; apply
  end;
  subst.


Lemma inv_recon_nn: forall ls et n n0,
    reconstruct_evP (evc ls et) (nnc n n0) ->
```

Fig. 1: CoqDog user interface, showing multi-shot proof refinement.

developers, for example providing a "tactic per lemma" without being asked, see Figure 10.

We were eventually able to submit more than 1,400 lines of lemmas to GPT-4, comprising some 58 lemmas, and saw no meaningful degradation in the quality of the output as the proof files got longer. Along the way, we discovered that not all LLMs were created equal in this regards; particularly, GPT-4 worked significantly better on the Coq proof generation task than any other model that we tried, including its predecessor GPT-3.5. (We were later able to narrow the gap between the two when we were given permission to utilize GPT-3.5 in "multi-shot" mode.) Additionally, we found that while our time on GPT-4 was restricted more than other LLMs, it was so much more efficient at producing valid proofs that it was worthwhile just to be a bit more circumspect in our inputs in order to access the higher-quality GPT-4 output.

Despite our early successes, as we pushed CoqDog to do more, we (unsurprisingly) encountered scalability and usability issues. In the sections that follow, we discuss two research problems identified during the development and evaluation of the CoqDog system, namely LLM Contextual Prompt Constraints and the Measurement of Proof Repair Quality. We outline our methodologies for addressing each of these challenges, present our key findings, and discuss current limitations as well as future work.

6 LLM Contextual Prompt Constraints Problem

In this section, we present our approach to address a challenging problem that we faced while evaluating the CoqDog system, originating from the prompt contextual constraints of LLM models, further exacerbated by the limitations of these models in logically classifying data. Such limitations may compromise the

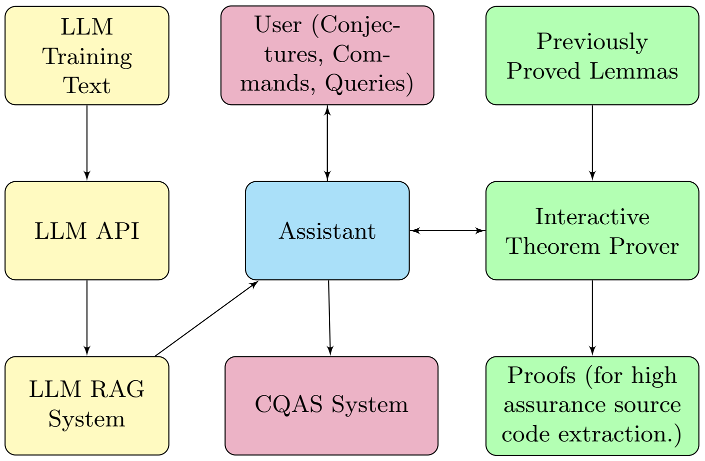

LLM Training Text
User (Conjectures, Commands, Queries)
Previously Proved Lemmas
LLM API
Assistant
Interactive Theorem Prover
LLM RAG System
CQAS System
Proofs (for high assurance source code extraction.)

Fig. 2: CoqDog Copilot workflow comprises several components. The user inputs lemmas and queries for proof, along with strategic proof hints. The assistant retrieves relevant data from training material and uses its customized RAG system for the Copland proofbase. CoqDog then generates proof candidates, which the user submits to the theorem prover. The theorem prover attempts to verify the lemmas using the generated proof candidates, providing feedback to the user and the assistant until a correct proof is achieved. The assistant sends conversations to the CQAS system for quality evaluation and visualizations.

efficacy of the copilot in realistic scenarios where the system necessitates the user to provide sufficient and logically related context for successful proof.

Initially, it was not overly challenging to equip CoqDog copilot with sufficient context in line with the constraints of the prompts during dialogues, particularly when the copilot prompted the user to provide specific definitions to proceed. However, the complexity of this issue escalates when a number of relevant lemmas need to be provided in the context to prove a theorem, where said lemmas are scattered in a large proofbase, a proofbase that is too large to provide as context.

Lemma and Proof Recommendation System Based on LLM RAG To mitigate these substantial challenges in the Copland case study, which encompasses a repository of approximately $\approx 21,000$ lines of Coq code, we explored the integration of text-embedding and augmented data retrieval. This was coupled with the development of a semantic search to implement a customized *Lemma and Proof* recommendation system for the Copland case study, as depicted in Figure 3.

We designed this system to assist CoqDog in offering logically sequenced, semantically, and contextually relevant lemmas and proofs tailored to the target lemmas as per user requests. This is implemented as a Retrieval-Augmented

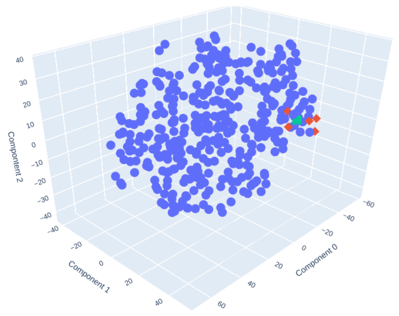


Fig. 3: Copland's latent vector space (3D) and CoqDog recommendations.

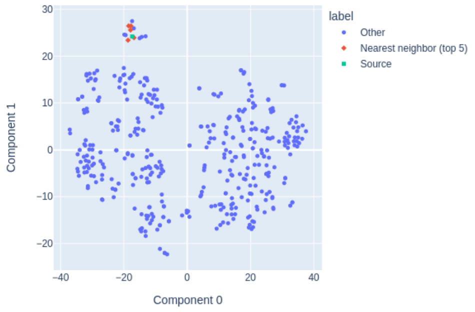


Fig. 4: Sampling Copland's 2D latent space using cosine similarity classification.

Generation (RAG) system supported by a ranking system that prevents unsound cycles in the reasoning recommendation process.

The system parses Coq files and extracts lemmas and proofs from the repository. These elements are indexed in a CSV file that maintains their order as presented within the Coq files. The Coq files adhere to the order specified in the `_CoqProject` file. Thus, the CoqDog ranking system is designed to help the recommendation system avoid import chain cycles. If file f_1 requires file f_2, then f_1 will have a higher rank. Additionally, it indexes the lemmas such that if lemma 1's proof calls lemma 2, then lemma 1 will have a higher index. We then use the `GPT text-embedding-ada-002` model, which transforms them into vector representations, storing the lemmas, proofs, and their corresponding vectors in a persistent data store. The CoqDog system then processes the user-input source lemma for verification and invokes the recommendation system, which transforms the lemma into a vector. We refer to the space of all these vectors by *Copland's latent vector space*, Figure 3. Then the system indexes the lemma into the data store if it does not exist already. The recommendation system employs the cosine distance function to classify the vectors, Figure 4, and identifies the k-nearest neighbors, Figure 3, while filtering out lemmas with greater or equal indices to prevent logical circularity in its recommendations. The CoqDog recommendations classification approach takes into consideration **topi**cal similarities that are **logical**ly ordered; hence, we describe it as a **topi-logical** classification.

Observe that our novel topi-logical recommendation system design addresses a problem with cosine similarities: while they provide a topological neighborhood of the target lemma efficiently, they do not account for the logical or functional similarities of the neighbors in the context of theorem proving.

Finally, we engineered CoqDog's user prompt to send a request to use the system's recommendations to generate a proof candidate for the source lemma using `GPT4` when the context size is less than `8000` tokens or `GPT3.5-turbo` when the context size is greater than `8000` but less than or equal to `16000` tokens. These boundaries adhere to our OpenAI account's context sizes for each model at the time of writing this paper. Note that when this research was being performed,

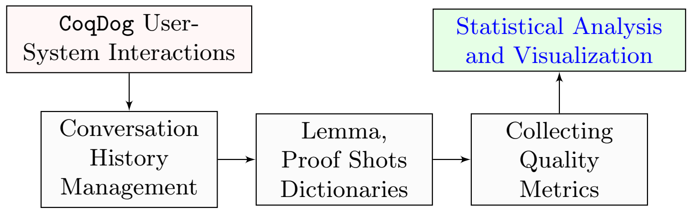

CoqDog User-System Interactions
Statistical Analysis and Visualization
Conversation History Management
Lemma, Proof Shots Dictionaries
Collecting Quality Metrics

Fig. 5: CoqDog Conversation Quality Assessment System (CQAS) workflow.

GPT-4 Turbo did not exist, so context size limitations were a significant issue and it will continue to be so for very large libraries.

7 Proof Repair Quality Assessment Problem

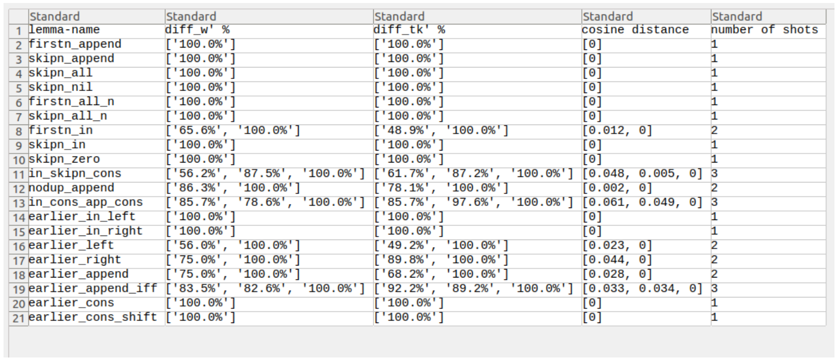
The conventional method for evaluating a text generated by Large Language Models involves token comparisons to a pre-defined *golden example* [11, 13, 14], followed by accuracy calculation. However, in theorem proving, it is crucial to acknowledge that syntactically different correct proofs of the same theorem are equally valid; as a result, this approach to accuracy assessment may lead to significant undervaluation of the model, prematurely leading the machine learning researcher towards far more expensive and elusive formal proof training data sets to fine-tune the models or reduce the creativity parameters of the models, which would be counterproductive.

The complexity of this challenge escalates when dealing with conversational multi-shot prompting, where the overall quality of the conversation's user prompts impacts output assessment [11, 14]. The scarcity of formal proofs for similar lemmas exacerbates the challenge even further, making the development of practical proof repair prompts and conversation quality assessment methods an urgent and compelling problem to solve. To address this open problem for the CoqDog copilot in the context of the Copland case study, we developed a Conversation Quality Assessment System (CQAS) for CoqDog copilot, illustrated in Figure 5.

7.1 CoqDog Conversation Quality Assessment System (CQAS)

To build CQAS, we have equipped CoqDog with a conversation history management functionality, enabling users to save conversations as machine-readable JSON files by the CQAS system. The CQAS parses the targeted lemmas and their proof shots lists and organizes them into JSON dictionaries of the format `{lemma-name: value, proof-shots: [proof-shot_1, proof-shot_-2, ..., proof-shot_n]}`, where `proof-shot_n` is typically the final correct proof verified by Coq. We refer to these as the `proof-shots` dictionaries.

Subsequently, the CQAS collects and statistically analyzes a set of novel proof repair metrics from CoqDog `proof-shots` dictionaries at three levels:



	Standard	Standard	Standard	Standard	Standard
1	lemma-name	diff_w' %	diff_tk' %	cosine distance	number of shots
2	firstn_append	['100.0%']	['100.0%']	[0]	1
3	skipn_append	['100.0%']	['100.0%']	[0]	1
4	skipn_all	['100.0%']	['100.0%']	[0]	1
5	skipn_nil	['100.0%']	['100.0%']	[0]	1
6	firstn_all_n	['100.0%']	['100.0%']	[0]	1
7	skipn_all_n	['100.0%']	['100.0%']	[0]	1
8	firstn_in	['65.6%', '100.0%']	['48.9%', '100.0%']	[0.012, 0]	2
9	skipn_in	['100.0%']	['100.0%']	[0]	1
10	skipn_zero	['100.0%']	['100.0%']	[0]	1
11	in_skipn_cons	['56.2%', '87.5%', '100.0%']	['61.7%', '87.2%', '100.0%']	[0.048, 0.005, 0]	3
12	nodup_append	['86.3%', '100.0%']	['78.1%', '100.0%']	[0.002, 0]	2
13	in_cons_app_cons	['85.7%', '78.6%', '100.0%']	['85.7%', '97.6%', '100.0%']	[0.061, 0.049, 0]	3
14	earlier_in_left	['100.0%']	['100.0%']	[0]	1
15	earlier_in_right	['100.0%']	['100.0%']	[0]	1
16	earlier_left	['56.0%', '100.0%']	['49.2%', '100.0%']	[0.023, 0]	2
17	earlier_right	['75.0%', '100.0%']	['89.8%', '100.0%']	[0.044, 0]	2
18	earlier_append	['75.0%', '100.0%']	['68.2%', '100.0%']	[0.028, 0]	2
19	earlier_append_iff	['83.5%', '82.6%', '100.0%']	['92.2%', '89.2%', '100.0%']	[0.033, 0.034, 0]	3
20	earlier_cons	['100.0%']	['100.0%']	[0]	1
21	earlier_cons_shift	['100.0%']	['100.0%']	[0]	1

Fig. 6: CQAS Metrics Calculation for MoreLists Conversation.

- Lemma — also referred to as dialogue. Corresponds to one `proof-shot` dictionary and represents the individual unit within a CQAS proof repair conversation analysis.
- Conversation — the whole sequence of dialogues in a JSON conversation file, typically representing a complete interaction for a sequence of lemmas.
- Sample Space — an analysis of all conversation files in a given sample space, combining insights from multiple conversations.

CQAS reports these metrics in CSV files and visualizes the statistical analysis for the user, as seen in Figure 7 and Appendix A Figure 8. Specifically, the CQAS methodology tries to practically correlate CoqDog dialogues, conversations, and sample spaces with statistical histograms to understand the distributions of selected metrics. This method allows us to practically differentiate between higher and lower-quality conversations while evaluating CoqDog on the Copland case study (as in Figure 7). Additionally, CQAS employs novel mathematical modeling to discern trends within these metrics at the dialogue level, offering further insights into the quality of the hints and prompts provided by users during the interactions with the CoqDog system.

7.2 CQAS Proof Repair Metrics

We designed CQAS to calculate four metrics for each proof-shot dictionary and to present it in CSV tables, as shown in Figure 6. [3] Our metrics are defined to compare the proof candidate in a dialogue with a *relative golden example*, specifically `proof-shot_n`. The metrics are correlated with the user's proof repair efforts to achieve a correct proof. Furthermore, they are designed to help us evaluate the conversation quality in terms of statistical analysis over user prompts. This allows us to vet the best prompting strategies and improve our interactions over time. Next, we illustrate each proof repair metric calculated by CQAS.

[3] Tables for the other clusters are provided in Appendix A.

Number of shots The length of the `proof-shots` list is an important proof repair metric, referred to as *number of shots*. We use this metric to monitor the number of repair shots required in the dialogue to transform an initial proof candidate into a valid proof.

This count of shots serves as one of the primary metrics for CQAS to statistically infer the quality of interactions with the CoqDog system from two perspectives: automation and success rate. The lower the number, the greater the automation, indicating less human effort is needed. Similarly, a smaller number indicates a higher success rate for the model in a given dialogue.

Furthermore, this metric allows for the convenient calculation of median, mean, and other central statistical measures at both conversation and sample space levels, as shown in Appendix A Figure 9.

This aids in statistically inferring the quality of one or more CoqDog conversations as well as the sample space, as we will demonstrate in Section 8, Figure 7 and Appendix A Figure 8.

On the flip side, although the `number of shots` metric is significantly helpful in the aforementioned cases, it only provides the number of shots required to achieve a `100%` correct proof without offering further details about the other proof candidates. This may obscure the impact of individual proof repair hints within the dialogue.

7.3 Proof Repair Oscillation Metrics

Mitigating the limitations of the `number of shots` metric proved to be a challenging task. To address this, we had to define a new set of metrics, each aimed at measuring a different aspect of the proof repair process. Specifically, users may desire to modify the initial proof candidate by removing existing steps, adding new ones, or even adjusting their alignments. For instance, it is crucial to note that while a modification to the prompt may improve the output for one input, it could potentially degrade performance for others [14].

Our approach, leveraging a Coq proof checker to pinpoint hallucinations, offers greater flexibility and precludes the need for more strict string matching methods as outlined in [11, 13].

In the sequel, we illustrate our CQAS's novel customized metrics, which are designed to measure and analyze the modifications of the proof repair process within a dialogue.

Words Count. This metric was initially defined by a ratio between the number of words `nw` of each `proof_shot_i` and `proof_shot_n` the relative golden example.

Specifically, it is calculated as:

$$\textit{diff\_ w} = \frac{\texttt{nw in proof\_shot\_i} - \texttt{nw in proof\_shot\_n}}{\texttt{nw in proof\_shot\_n}} \tag{1}$$

The definition of $\mathit{diff\_w}$ responds to the addition and removal operations in a proof repair process, and the resulting value can vary either positively or negatively.

Furthermore, the value $\mathit{diff\_w}' = |1 - |\mathit{diff\_w}||$ can be greater than one or less than one. It must be 1 when the correct relative golden example is reached since $|\mathit{diff\_w}|$ will be zero.

Note that the absolute value $|\mathit{diff\_w}|$ can provide insight into the magnitude of the change occurring at each proof repair shot, regardless of the direction. In our study, we aimed to see this value approaching zero during the proof repair process or, equivalently, the value $\mathit{diff\_w}'$ approaching one.

In CQAS CSV files such as shown in Figure 6, we report $\mathit{diff\_w}'$ for the user, as it can be interpreted more easily.

For simplicity, we define a word as any sequence of characters except for white spaces or new lines. The CQAS system will then split the proof shot strings using white spaces to identify the number of words for this metric.

Tokens Count. The `Word Count` metric has inherent limitations, particularly in that it does not respond to changes at the sub-word level during the proof repair process. Therefore, we employ the `Token Count` metric. We used `gpt-4` model tokenizer, offered by OpenAI library, to find the number of tokens, denoted as `tk_n`.

We define this metric analogously to the previous one but using `tk_n` instead of the number of words.

$$\mathit{diff\_tk} = \frac{\texttt{tk\_n in proof\_shot\_i} - \texttt{tk\_n in proof\_shot\_n}}{\texttt{tk\_n in proof\_shot\_n}} \tag{2}$$

Similarly, we define $\mathit{diff\_tk}' = |1 - |\mathit{diff\_tk}||$ and report it in CSV file.

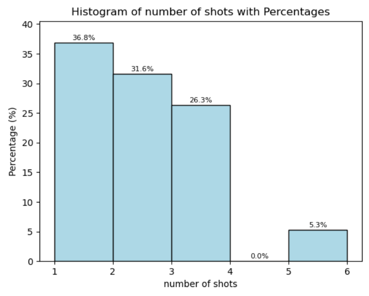
While this metric shares similarities with the previous one, it is more sensitive to textual changes. This characteristic enables it to account for sub-word changes. However, there is a cost associated, as using the tokenizer, while very inexpensive, is not free.

Cosine Distance. It should be noted that both the `Token Count` and `Word Count` metrics are sensitive to syntactic changes in the proof step. However, that alone may not be particularly helpful in providing detailed insight into the semantic changes during the proof repair process.

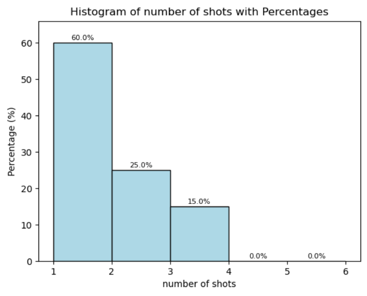
For this reason, we define the `Cosine Distance` metric. The CQAS system converts every `proof_shot_i` into a vector `V` using the `text-embedding-ada-002` and then computes the cosine distance between the proof shot and relative golden example `proof_shot_n`.

$$\texttt{Cosine Distance} = \texttt{Cosine\_dist(V(proof\_shot\_i),V(proof\_shot\_n))} \tag{3}$$

This metric can provide the user insights into semantic changes and is less sensitive to syntactic changes than the previous two metrics. In particular, the



(a) Lower quality conversation for MoreLists clusters (5 shot correctness).



(b) Higher quality conversation for MoreLists clusters (3 shot correctness).

Fig. 7: Novel CQAS visualized statistical distributions to describe conversation quality. The figure (7b)'s leftward skew signifies higher quality through more automation and a higher success rate. CQAS quality assessment at the conversation level compares results without (7a) and with (7b) well-structured generative proof strategies. The same sets of lemmas but different prompts by the same user led to significantly better results for the same model, demonstrating the impact of prompt structuring on performance, and it shows that CQAS's visualization system can detect this clearly.

metric is capable of detecting changes in the order of words and tokens in proof steps, such changes can elude detection by the `Word Count` and `Token Count` metrics. Ideally, this metric should be zero when the relative golden proof is reached. We aim to see it approach zero during the repair process as well.

8 Experimental Evaluation

The Copland repository, developed over many years, comprises about `21,000` Coq Lines of Code (LoC) and `213` lemmas. Its lemmas are classified into two categories: *Copland Reference Semantics* and *Copland Virtual Machine and Appraisal Semantics*. The CQAS categorizes these into *target lemmas* (`58`) and *library lemmas* (`155`). The library lemmas were indexed as explained in Section 6 while the target lemmas were used for assessment. CQAS topi-logically orders most of *Copland Virtual Machine and Appraisal Semantics* lemmas with higher indices. Therefore, due to logical dependencies, our current focus is on the *Copland Reference Semantics* lemmas for system assessment. This group's lemmas are spread across `13` Coq files, totaling around `4,927` LoC, with `1,400` LoC in target lemmas.

8.1 Key Results

We employed the CQAS methodology in our assessment, as described in Section 7. Specifically, the target lemmas were analyzed in three disjoint conversations. In the context of Copland's latent vector space, as seen in Figure 3 and Figure 4, each conversation may include one or more topi-logical clusters of lemmas. We summarize our main results as follows:

– CQAS has effectively enabled us to identify a novel approach to interacting with GPT-4 and GPT-3.5 models for theorem-proving purposes. Notably, the strategic inclusion of human-like, high-level hints to the proof repair prompts such as `use lemma1,...,lemma_n,` and `use tactics tac1,..., tac_n as appropriate` can significantly improve the quality of the proof candidates, as shown in Figure 7.
– Without additional fine-tuning of the backend models, our approach has helped us to achieve a success rate exceeding 60% for proof generation using the CoqDog GPT-4 option in a single 'shot'. Moreover, approximately 30% more lemmas were proven with a second 'shot' (resulting in about 90% overall correctness). With three 'shots', the overall proof correctness rate increases to 97%, as shown in Appendix A Figure 8d.

These results suggest that, without well-structured proof repair hints appended to the context prompts, users might not fully be able to leverage the potential of LLMs such as GPT-4 and GPT-3.5 models in formal reasoning.

8.2 Current Limitations and Future Work

While our approach was significantly helpful in harnessing the knowledge of the backend GPT models, extending their understanding without additional training, and achieving a high success rate within three attempts on the current sample, we plan to expand the coverage. Our future goal is to include other lemma groups and a larger code sample from the Copland code base. Furthermore, we believe that enhancing the system's ability with definition retrieval could automate the process further.

Additionally, while we found the lemma-proof recommendation's logical circularity solution to be sound, it may produce redundant recommendations in cases where the topological lemma neighborhoods of two separate targeted lemmas in the same conversation intersect in the Copland's latent vector space. This issue can be mitigated by implementing a redundancy filter. Another aspect for development is a voting system. This system could enable users to recommend hints to the CoqDog system, thereby enhancing its decision-making process. We also plan to adapt the technologies developed for our Coq assistant to other interactive theorem provers and SMT solvers, such as Isabelle, Lean, Verus, and the CBMC model checker.

9 Related Work

Our work draws on two distinct areas of research: high-assurance remote attestation, and the use of Large Language Models for Automated Reasoning.

9.1 Remote Attestation

Remote attestation [5, 7] is a technique for gathering and appraising evidence from a remote system to determine if that system adheres to some policy. Typically, remote attestation is applied to measurements gathered when software is loaded and started [26] or during system runtime [5,17,19]. Regardless, evidence is gathered with the expressed intent of appraising system behavior.

Runtime attestation is typically centered on execution of *attestation protocols* by *attestation managers*. Each protocol specifies measurements, cryptographic operations, and work dispatch to execute in a specific order on an attestation manager. Measurements gather evidence directly from their target. Measurements range in complexity from simple file hashes to contextual measurements describing kernel memory. Cryptographic operations are typically signing operations that guarantee both evidence integrity and authenticity. Dispatch operations make requests of other attestation managers, providing a mechanism for layered attestations [24] and scaling attestation to large systems.

Copland [23] originated as a domain specific language for representing attestation protocols for run-time, layered attestations [24]. Copland's formal semantics support construction of verified attestation tools and components. Copland has purpose built language support for coordinating and distributing work in highly flexible ways suitable for complex attestation algorithms. The same flexibility supports moving from attestation to the more general problem of systematically gathering different kinds of evidence that may represent test results, verification results, certification outcomes, or any other information useful for maintaining the measured system.

Because Copland protocols are data structures, they are first-class in our attestation development environments. Specifically, they support reasoning about and manipulating protocols. The beginnings of this capability are seen here, but significant work remains to approach maintaining protocols and the evidence they gather over an evolving system.

9.2 Using LLMs for Automated Reasoning

Researchers from OpenAI performed some of the earliest research on the use of LLMs for mechanical theorem proving in 2020 [22]. The OpenAI team utilized a GPT-3 base, but also produced model variants that had a higher mix of input from Github, arXiv Math, and Math StackExchange (the latter was associated with a 10% improvement in proof performance). The result was GPT-f, a proof assistant for Metamath. Metamath is a simple proof language targeted to mathematics [10]. GPT-f was able to achieve a peak Metamath proof success rate of 56%, and proved some 200 Metamath theorems. GPT-f also discovered a few shorter proofs for existing lemmas in the Metamath proofbase, proofs which the Metamath team later incorporated.

Other groups have studied proof generation and repair using LLMs. First *et al.* report a success rate of approximately 50% for a proof repair scenario for Isabelle/HOL [6], using Minerva [9], a large language model pretrained on a mathematics corpus. Minerva is based on Google's PaLM LLM [3]. Many of these same researchers investigated proving theorems in Coq using GPT-3.5 and GPT-4 [28], similar to our work. Their paper, however, focused more on diagnosing failed proof attempts when using the LLM rather than characterizing their success rate, so it is difficult to directly compare their results to ours.

Researchers have also utilized LLM technology on related problems in automated reasoning. LLMs have been used to discover program invariants, in particular loop invariants, a well-known difficult problem in code verification, with notable success [16, 27]. Researchers at Stanford and VMware Research are developing an approach for closed-loop verifiable code generation called Clover [25]. Clover considers three classes of artifacts created during development using the verification-enhanced programming language Dafny [8], namely the English language Specification for a given Dafny program, the Code in Dafny, which is interspersed with formal Annotations. These artifacts may be generated manually or automatically through the use of generative AI technology. Clover then performs a set of consistency checks amongst the artifacts, including: Dafny formal verification, to ensure that the code is functionally correct with respect to its annotations; reconstruction of the annotations from the code (nominally using LLM technology) to ensure that the annotations capture the full functionality of the code; and reconstruction of the Specification from the Annotations or the Code to determine Specification accuracy. On benchmark Dafny programs containing both correct and incorrect artifacts, the prototype Clover system accepts correct artifacts 75% of the time, and rejects 100% of incorrect artifacts.

10 Conclusion

In approximately four months of intense investigation, we were able to develop a very capable Coq proof repair capability using LLMs. In future work, we look to increase the number of lines of Coq input that can be submitted to the LLM system, continue to investigate novel "multi-shot" prompting scenarios that can rapidly refine the LLM output to produce a valid proof, research the LLMs capability to generate subsidiary lemmas on its own in order to lead to a proof, as well as explore novel ways that LLMs can be utilized to assist the process of high-assurance code extraction from theorem provers such as Coq to various programming languages. We also plan to adapt the technologies developed for our Coq assistant to other interactive theorem provers and SMT solvers, such as Isabelle, Lean, Verus, and the CBMC model checker.

11 Acknowledgments

This work was funded by DARPA contract HR00111890001. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

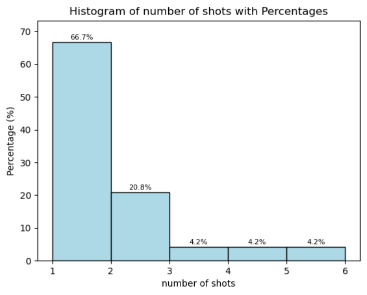
Remark 2. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

References

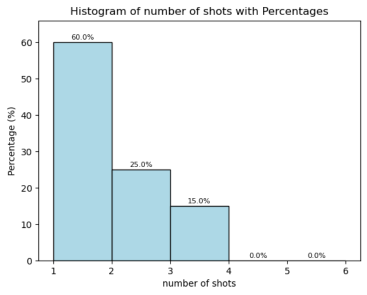
1. Amundson, I., Cofer, D.: Resolute assurance arguments for cyber assured systems engineering. In: Design Automation for Cyber-Physical Systems and Internet of Things (DESTION 2021) (May 2021)
2. Belt, J., Hatcliff, J., Robby, Shackleton, J., Carciofini, J., Carpenter, T., Mercer, E., Amundson, I., Babar, J., Cofer, D., Hardin, D., Hoech, K., Slind, K., Kuz, I., Mcleod, K.: Model-driven development for the seL4 microkernel using the HAMR framework. Journal of Systems Architecture **134**, 102789 (2023). https://doi.org/10.1016/j.sysarc.2022.102789, https://www.sciencedirect.com/science/article/pii/S1383762122002740
3. Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H.W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A.M., Pillai, T.S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., Fiedel, N.: PaLM: Scaling language modeling with pathways (2022), https://arxiv.org/pdf/2204.02311.pdf
4. Cofer, D., Amundson, I., Babar, J., Hardin, D., Slind, K., Alexander, P., Hatcliff, J., Robby, Klein, G., Lewis, C., Mercer, E., Shackleton, J.: Cyber assured systems engineering at scale. In: IEEE Security & Privacy. pp. 52–64 (May/June 2022). https://doi.org/10.1109/MSEC.2022.3151733
5. Coker, G., Guttman, J., Loscocco, P., Herzog, A., Millen, J., O'Hanlon, B., Ramsdell, J., Segall, A., Sheehy, J., Sniffen, B.: Principles of remote attestation. International Journal of Information Security **10**(2), 63–81 (June 2011)
6. First, E., Rabe, M.N., Ringer, T., Brun, Y.: Baldur: Whole-proof generation and repair with large language models (2023), https://arxiv.org/pdf/2303.04910.pdf
7. Haldar, V., Chandra, D., Franz, M.: Semantic remote attestation – a virtual machine directed approach to trusted computing. In: Proceedings of the Third Virtual Machine Research and Technology Symposium. San Jose, CA (May 2004)
8. Leino, K.R.M.: Developing verified programs with Dafny. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 1488–1490. ICSE '13, IEEE Press, Piscataway, NJ, USA (2013), http://dl.acm.org/citation.cfm?id=2486788.2487050
9. Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T., Wu, Y., Neyshabur, B., Gur-Ari, G., Misra, V.: Solving quantitative reasoning problems with language models (2022), https://arxiv.org/pdf/2206.14858.pdf
10. Megill, N., Wheeler, D.A.: Metamath: A computer language for mathematical proofs (2019), https://us.metamath.org/downloads/metamath.pdf

11. OpenAI: Evaluation templates (2023), https://github.com/openai/evals/blob/main/docs/eval-templates.md, accessed on: 9-12-2023
12. OpenAI: GPT-4 Technical Report (2023), https://arxiv.org/pdf/2303.08774.pdf
13. OpenAI: Legacy fine-tuning guide. https://platform.openai.com/docs/guides/legacy-fine-tuning (2023), accessed: 9-12-2023
14. OpenAI: Prompt engineering strategies (2023), https://platform.openai.com/docs/guides/prompt-engineering/strategy-use-external-tools, accessed on: 9-12-2023
15. Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., Karri, R.: Asleep at the keyboard? assessing the security of GitHub Copilot's code contributions. In: 2022 IEEE Symposium on Security and Privacy. pp. 754–768 (2022). https://doi.org/10.1109/SP46214.2022.9833571
16. Pei, K., Bieber, D., Shi, K., Sutton, C., Yin, P.: Can large language models reason about program invariants? In: Krause, A., Brunskill, E., Cho, K., Englehardt, B., Sabato, S., Scarlett, J. (eds.) Proceedings of the 40th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 202, pp. 27496–27520. PMLR (July 2023), https://proceedings.mlr.press/v202/pei23a/pei23a.html
17. Pendergrass, J.A., Helble, S., Clemens, J., Loscocco, P.: Maat: A platform service for measurement and attestation. arXiv preprint arXiv:1709.10147 (2017)
18. Perry, N., Srivastava, M., Kumar, D., Boneh, D.: Do users write more insecure code with AI assistants? (2022), https://arxiv.org/pdf/2211.03622.pdf
19. Petz, A., Alexander, P.: An Infrastructure for Faithful Execution of Remote Attestation Protocols. Innovations in Systems and Software Engineering (2022)
20. Petz, A., Alexander, P.: An infrastructure for faithful execution of remote attestation protocols. In: Proceedings of the 13th NASA Formal Methods Symposium (NFM 2021) (May 2021)
21. Petz, A., Jurgensen, G., Alexander, P.: Design and formal verification of a Copland-based attestation protocol. In: ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE 2021) (November 2021)
22. Polu, S., Sutskever, I.: Generative language modeling for automated theorem proving (2020), https://arxiv.org/pdf/2009.03393.pdf
23. Ramsdell, J., Rowe, P.D., Alexander, P., Helble, S., Loscocco, P., Pendergrass, J.A., Petz, A.: Orchestrating layered attestations. In: Principles of Security and Trust (POST'19). Prague, Czech Republic (April 8-11 2019)
24. Rowe, P.D.: Bundling Evidence for Layered Attestation. In: Trust and Trustworthy Computing, pp. 119–139. Springer International Publishing, Cham (Aug 2016)
25. Sun, C., Sheng, Y., Padon, O., Barrett, C.: Clover: Closed-loop verifiable code generation (2024), https://arxiv.org/pdf/2310.17807.pdf
26. Trusted Computing Group: TCG TPM Specification. Trusted Computing Group, 3885 SW 153rd Drive, Beaverton, OR 97006, version 1.2 revision 103 edn. (July 2007), https://www.trustedcomputinggroup.org/resources/tpm_main_specification/
27. Wu, H., Barrett, C., Narodytska, N.: Lemur: Integrating large language models in automated program verification (2023), https://arxiv.org/pdf/2310.04870.pdf
28. Zhang, S.D., First, E., Ringer, T.: Getting more out of large language models for proofs (2023), https://arxiv.org/pdf/2305.04369.pdf

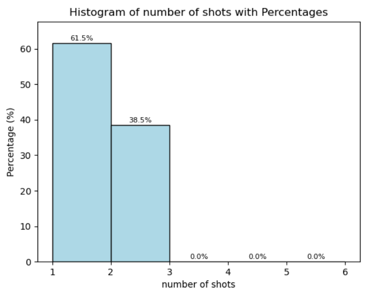
A CQAS Metrics Tables, Statistical Evaluations, and Visualizations



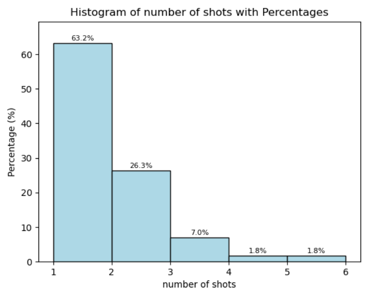
(a) Conversational quality assessment for LTS cluster.



(b) Conversational quality assessment for MoreLists cluster.

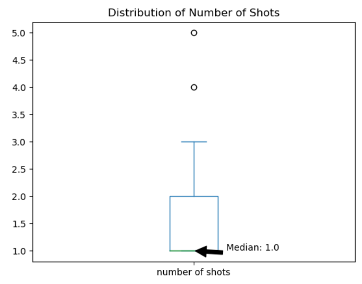


(c) Conversational quality assessment for MonadLaws cluster.

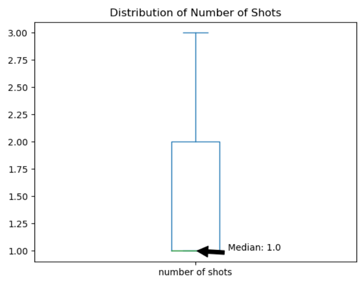


(d) Overall quality assessment using all clusters combined.

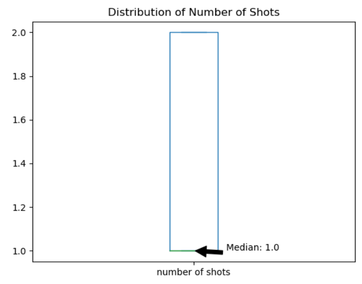
Fig. 8: The figure shows an example of CQAS combined quality assessment for three clusters.



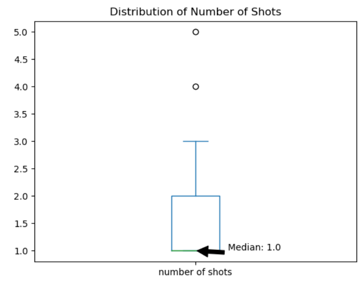
(a) CQAS central tendency values for LTS clusters.



(b) CQAS central tendency for MoreLists clusters.

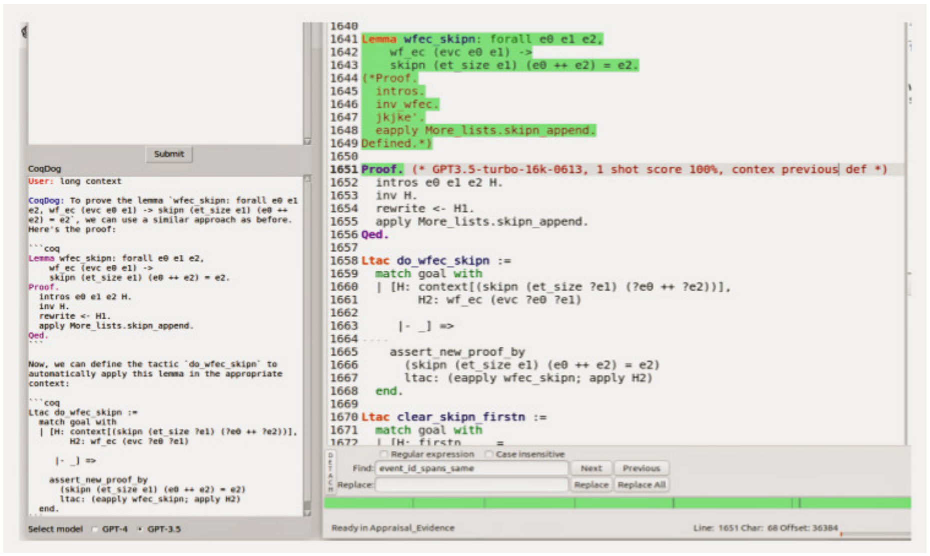


(c) CQAS central tendency for MonadLaws cluster.



(d) Overall quality assessment for the sample space using all clusters combined.

Fig. 9: CQAS quality assessment measures of central tendency at the sample space level.



~~~
User: long context

CoqDog: To prove the lemma `wfec_skipn: forall e0 e1
e2, wf_ec (evc e0 e1) -> skipn (et_size e1) (e0 ++
e2) = e2`, we can use a similar approach as before.
Here's the proof:

```coq
Lemma wfec_skipn: forall e0 e1 e2,
    wf_ec (evc e0 e1) ->
    skipn (et_size e1) (e0 ++ e2) = e2.
Proof.
  intros e0 e1 e2 H.
  inv H.
  rewrite <- H1.
  apply More_lists.skipn_append.
Qed.
~~~

Now, we can define the tactic `do_wfec_skipn` to
automatically apply this lemma in the appropriate
context:

```coq
Ltac do_wfec_skipn :=
  match goal with
  | [H: context[(skipn (et_size ?e1) (?e0 ++ ?e2))],
        H2: wf_ec (evc ?e0 ?e1)

     |- _] =>

    assert_new_proof_by
      (skipn (et_size e1) (e0 ++ e2) = e2)
      ltac: (eapply wfec_skipn; apply H2)
  end.
```

```

Lemma wfec_skipn: forall e0 e1 e2,
    wf_ec (evc e0 e1) ->
    skipn (et_size e1) (e0 ++ e2) = e2.
(*Proof.
  intros.
  inv wfec.
  jkjke'.
  eapply More_lists.skipn_append.
Defined.*)

Proof. (* GPT3.5-turbo-16k-0613, 1 shot score 100%, contex previous def *)
  intros e0 e1 e2 H.
  inv H.
  rewrite <- H1.
  apply More_lists.skipn_append.
Qed.

Ltac do_wfec_skipn :=
  match goal with
  | [H: context[(skipn (et_size ?e1) (?e0 ++ ?e2))],
        H2: wf_ec (evc ?e0 ?e1)

     |- _] =>

    assert_new_proof_by
      (skipn (et_size e1) (e0 ++ e2) = e2)
      ltac: (eapply wfec_skipn; apply H2)
  end.

Ltac clear_skipn_firstn :=
  match goal with
  | [H: firstn      =
```

Fig. 10: The bottom left corner shows that CoqDog can generate tactics on the fly based on Chain-of-Thought conversational pattern detection with the user.

lemma-name	diff_w' %	diff_tk' %	cosine distance	number of shots
firstn_append	['100.0%']	['100.0%']	[0]	1
skipn_append	['100.0%']	['100.0%']	[0]	1
skipn_all	['100.0%']	['100.0%']	[0]	1
skipn_nil	['100.0%']	['100.0%']	[0]	1
firstn_all_n	['100.0%']	['100.0%']	[0]	1
skipn_all_n	['100.0%']	['100.0%']	[0]	1
firstn_in	['65.6%', '100.0%']	['48.9%', '100.0%']	[0.012, 0]	2
skipn_in	['100.0%']	['100.0%']	[0]	1
skipn_zero	['100.0%']	['100.0%']	[0]	1
in_skipn_cons	['56.2%', '87.5%', '100.0%']	['61.7%', '87.2%', '100.0%']	[0.048, 0.005, 0]	3
nodup_append	['86.3%', '100.0%']	['78.4%', '100.0%']	[0.002, 0]	2
in_cons_app_cons	['85.7%', '78.6%', '100.0%']	['85.7%', '97.6%', '100.0%']	[0.061, 0.049, 0]	3
earlier_in_left	['100.0%']	['100.0%']	[0]	1
earlier_in_right	['100.0%']	['100.0%']	[0]	1
earlier_left	['56.0%', '100.0%']	['49.2%', '100.0%']	[0.023, 0]	2
earlier_right	['75.0%', '100.0%']	['89.8%', '100.0%']	[0.044, 0]	2
earlier_append	['75.0%', '100.0%']	['68.2%', '100.0%']	[0.028, 0]	2
earlier_append_iff	['83.5%', '82.6%', '100.0%']	['92.2%', '89.2%', '100.0%']	[0.033, 0.034, 0]	3
earlier_cons	['100.0%']	['100.0%']	[0]	1
earlier_cons_shift	['100.0%']	['100.0%']	[0]	1

Table 1: CoqDog CQAS Metrics Calculation for MoreLists Cluster

lemma-name	**diff_w'%**	**diff_tk'%**	**cosine distance**	**number of shots**
monad_left_id	['100.0%']	['100.0%']	[0]	1
monad_right_id	['95.5%', '100.0%']	['97.9%', '100.0%']	[0.032, 0]	2
monad_right_id_ (S A)	['100.0%']	['100.0%']	[0]	1
monad_right_id_ (S A B C)	['99.0%', '100.0%']	['99.6%', '100.0%']	[0.008, 0]	2
monad_comp_ (S A B C)	['86.6%', '100.0%']	['91.2%', '100.0%']	[0.004, 0]	2
monad_get_get (S A)	['100.0%']	['100.0%']	[0]	1
monad_get_put (A)	['100.0%']	['100.0%']	[0]	1
monad_put_get (S A)	['100.0%']	['100.0%']	[0]	1
monad_put (S)	['85.7%', '100.0%']	['88.0%', '100.0%']	[0.013, 0]	2
fa_fa (A B)	['13.3%', '100.0%']	['17.6%', '100.0%']	[0.145, 0]	2
hlhl (A B)	['100.0%']	['100.0%']	[0]	1
hghg (A B)	['100.0%']	['100.0%']	[0]	1

Table 2: CoqDog CQAS Metrics Calculation for MonadLaws Cluster.

lemma-name	**diff_w' %**	**diff_tk' %**	**cosine distance**	**number of shots**
step_pl_eq	['100.0%']	[0]	0	1
step_seval	['100.0%']	[0]	0	1
star_transitive	['81.8%', '100.0%']	[0.015, 0]	0.015, 0	2
star_star	['100.0%']	[0]	0	1
star_lstar	['100.0%']	[0]	0	1
star_seval	['83.3%', '100.0%']	[0.015, 0]	0.015, 0	2
steps_preserves_eval	['100.0%']	[0]	0	1
star_strem	['100.0%']	[0]	0	1
star_stls	['100.0%']	[0]	0	1
star_stbsl	['100.0%']	[0]	0	1
star_stbsr	['100.0%']	[0]	0	1
star_stls	['100.0%']	[0.013, 0]	0.013, 0	2
star_strem	['100.0%']	[0.011, 0]	0.011, 0	2
star_stbsl	['100.0%']	[0.009, 0.01, 0.009, 0.008, 0]	0.009, 0.01, 0.009, 0.008, 0	5
star_stbp_exists	['70.0%', '100.0%']	[0.048, 0]	0.048, 0	2
star_stbparr	['82.7%', '82.7%', '71.9%', '80.0%', '100.0%']	[0.009, 0.01, 0.009, 0.008, 0]	0.009, 0.01, 0.009, 0.008, 0	5
star_stbp	['100.0%']	[0.008, 0.002, 0.008, 0]	0.008, 0.002, 0.008, 0	1
correct_path_exists	['82.7%', '82.7%', '71.9%', '80.0%', '100.0%']	[0.009, 0.01, 0.009, 0.008, 0]	0.009, 0.01, 0.009, 0.008, 0	5
never_stuck	['100.0%']	[0]	0	1
nstar_transitive	['100.0%']	[0.008, 0.002, 0.008, 0]	0.008, 0.002, 0.008, 0	4
nstar_star	['87.5%', '87.5%', '87.2%', '87.2%', '100.0%']	[0.008, 0.002, 0.008, 0]	0.008, 0.002, 0.008, 0	4
nstar_nstar	['90.3%', '90.3%', '100.0%']	[0.007, 0.007, 0]	0.007, 0.007, 0	3

Table 3: CoqDog CQAS Metrics Calculation for LTS Cluster.