Building a Large Proof Repair Dataset

Author One
Affiliation One

Author Two
Affiliation Two

Author Three
Affiliation Three

Talia Ringer
University of Illinois Urbana-Champaign

Abstract

We introduce a new, large proof-repair dataset and benchmark suite for the Coq proof assistant. The dataset is made up of Git commits from hundreds of open-source projects with old and new versions of definitions and proofs aligned across commits. Building this dataset was a significant undertaking, highlighting a number of challenges and gaps in existing infrastructure. We discuss these challenges and gaps, and we provide recommendations for how the proof assistant community can address them. Our hope is to make it easier to build datasets and benchmark suites so that machine-learning tools for proofs will move to target the tasks that matter most and do so equitably across proof assistants.

2012 ACM Subject Classification Computing methodologies → Machine learning; Software and its engineering → Software maintenance tools; Security and privacy → Logic and verification

Keywords and phrases TODO: TODO keywords

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Supplementary Material TODO URL to any code or data we can share

Funding This research was developed with funding from the Defense Advanced Research Projects Agency. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

1 Introduction

Machine learning is coming for proofs. Recent years have seen a surge in interest in machine learning for proofs—one reflected by the many recent research venues [4, 5, 6], papers [21, 34, 27], tools [7, 12, 25], industrial research groups [36, 2], and funding opportunities [1, 3] centering or prominently featuring machine learning for proofs. The surge in interest blurs the line between proofs and data so that any proof development, once released, may itself become data to improve proof automation for future proof developments.

We in the proof engineering community have agency in how this surge of interest plays out. We can close our eyes and hope that we are sufficiently different from the many fields that machine learning has changed. Or we can embrace that change and adapt; make sure that our voices are heard. We can develop datasets and benchmark suites that steer the machine-learning community toward the tasks that matter most. We can build infrastructure that makes it easy to develop those datasets and benchmark suites, or to work on those tasks. And we can build evaluation methodologies that measure success on those tasks in ways that truly matter, so that state-of-the-art results on benchmarks will transfer smoothly to real-world improvements in proof automation.

This paper takes a step in that direction. In particular, it presents a dataset and benchmark suite for an important proof automation task in Coq. The task of choice is *proof repair* [38]: the automatic repair of proofs in response to breaking changes in programs or specifications. Proof repair is a useful task for which data is scarce and challenging to collect [41]. This paper presents a new proof repair dataset and benchmark suite while also highlighting the challenges involved in building it, with an emphasis on how the proof engineering community can adapt to those challenges as machine learning becomes increasingly relevant.

Its contributions are the following:

1. a proof repair dataset and benchmark suite for Coq that is accessible to machine learning experts (Section 3),
2. tools for building and extracting information from Coq projects reusable for future dataset efforts (Section 4), and
3. a forward-facing discussion of the challenges we encountered in the course of this effort and how to overcome them (Section 5)

Our overarching goal is to build the infrastructure and proof assistant community support we need to steer the machine learning community toward the tasks that matter most before it is too late (Section 2).

2 Overview

A proof is a serialization of a truth, and assuming that our logical foundations are sound, truth is inviolable: what is true will always be true. This is great in isolation, but in practice, the software and hardware that form our proof's hypotheses about the world change over time, and so do the truths we need to prove. All software, even that which is formally verified, must receive maintenance to survive sustained contact with reality.

The necessity and difficulty of proof maintenance has been borne out empirically. A recent user study of eight intermediate through expert proof engineers showed that maintenance happened constantly for participating proof engineers during proof development [41], and that even experts sometimes gave up in the face of change [38].

Consider, for example, the change in Figure 1, in which a user study participant updated a lemma statement in response to a change in a dependency. As noted in the user study paper, this was part of a larger change, with 10 other definitions or lemma statements changing in analogous ways. Furthermore, this change broke at least five proofs, four of which the user study participant—an expert proof engineer—admitted or aborted rather than repair.

The ubiquity of maintenance and the challenges of repair have been largely neglected in machine-learning tools for proofs. Machine-learning tools for proofs have instead historically fixated on development tasks like predicting tactics [45, 12, **?**], synthesizing proofs from scratch [**?**, **?**, **?**], TODO: TOM: rephrase this re slack messages and automatically formalizing informal natural language statements [**?**, **?**, **?**].

If maintenance is so ubiquitous and repair is so challenging, what stands in the way of good machine-learning tools for proof repair? One of the central challenges is building a good dataset and benchmark suite, as datasets and benchmark suites can really drive results in machine learning for any domain. If the datasets and benchmark suites are not fit for maintenance tasks, the machine-learning community may neglect those tasks entirely, instead chasing state-of-the-art results on tasks for which existing benchmark suites suffice.

Existing datasets and benchmark suites are not sufficient for training and evaluating proof repair models. The main dataset of use in Coq is CoqGym [51], a collection of 124 proof developments in Coq. The CoqGym data for proof developments is static in that

```
  Lemma proc_rspec_crash_refines_op T (p : proc C_Op T)
    (rec : proc C_Op unit) spec (op : A_Op T) :
    (forall sA sC,
-    absr sA sC tt -> proc_rspec c_sem p rec (refine_spec spec sA)) ->
-   (forall sA sC, absr sA sC tt -> (spec sA).(pre)) ->
+    absr sA (Val sC tt) -> proc_rspec c_sem p rec (refine_spec spec sA)) ->
+   (forall sA sC, absr sA (Val sC tt) -> (spec sA).(pre)) ->
    (forall sA sC sA' v,
-    absr sA' sC tt ->
+    absr sA' (Val sC tt) ->
     (spec sA).(post) sA' v -> (op_spec a_sem op sA).(post) sA' v) ->
    (forall sA sC sA' v,
-    absr sA sC tt ->
+    absr sA (Val sC tt) ->
     (spec sA).(alternate) sA' v -> (op_spec a_sem op sA).(alternate) sA' v) ->
    crash_refines absr c_sem p rec (a_sem.(step) op)
      (a_sem.(crash_step) + (a_sem.(step) op;; a_sem.(crash_step))).
```

Figure 1 Changes made to a lemma by a participant in a recent user study of proof engineers, from the user study paper [41].

it includes the projects frozen in time, rather than the histories or changes that would be needed to train a model for proof repair. In line with this, the CoqGym benchmarks measure success on a proof synthesis task in terms of the number of proofs synthesized in the suite for a given set of theorems. The data from the REPLica [41] user study of Coq proof engineers, in contrast, includes very granular changes useful for repair, but is far too small in size to use to train dedicated proof repair models.

Our experiences interacting with machine-learning experts and building datasets ourselves suggest that the choice of datasets and benchmark suites for a domain is not driven solely by what is likely to be useful—it is also driven by barriers imposed by infrastructure, lack of domain expertise, or social factors. Here are three examples of these barriers:

1. In building a model for a brand new task with an industrial team, one of the authors—a Coq expert—chose Isabelle/HOL over Coq because of the utility of the Archive of Formal Proofs. They were unable to build a dataset for this task in Coq on a reasonable timeline because Coq lacked a **centralized archive** with **well-documented versioned histories**.
2. One of the authors witnessed several machine-learning experts across industry and academia erase meaningful information from data or make simplifying decisions about data and metrics for success that may corrupt results because of the difficulties in automating **parsing** and **building** Isabelle/HOL proof developments, and in **checking** proofs. These machine-learning experts largely preferred to treat proofs as plaintext data, and to interact with the proof assistant as little as possible.
3. The REPLica dataset of atomic edits in Coq was fundamentally limited in size by the difficulties in **recruiting enough participants** in the user study [41].

In this paper, we overcome a number of these challenges and build a large proof repair dataset for Coq. We also discuss the barriers we do encounter, and we describe both how we overcome those barriers, and what kind of work the proof assistant community would need to put in to make it so that they cease to be barriers at all.

We find this especially prudent given that the danger of chasing benchmarks that may not transfer to real life workflows has been realized quite dramatically in other domains—from incorrect patches to programs [35] to incorrect clinical interpretation of x-ray results [53]. Furthermore, these challenges can influence not just the tasks that the machine-learning community chooses to tackle, but even the very proof assistants the machine-learning community chooses to build tools in support of. It is in the community's best interest to drive strong, practical results for useful tasks in a way that is equitable across proof assistants.

Our hopes are twofold. First, we hope that our dataset will be immediately useful for proof repair. Second, and perhaps more prudently, we hope our discussion of the challenges involved in building it will serve as a call for action to build better infrastructure. That way, we in the proof engineering community can ensure that machine-learning experts focus on the proof automation tasks that matter most to our community, that they measure success on those tasks in ways that are likely to transfer smoothly to real-world usefulness, and that they do so equitably across proof assistants.

3 A Proof Repair Dataset

The dataset and benchmark suite that we have assembled will be publicly available.[1] The task our dataset and benchmark suite focuses on is proof repair (Section 3.1). The data comprise aligned Git commits that correspond to existing changes in proof developments found on GitHub (Section 3.2). Success on the resulting benchmark is evaluated in terms of successful proof checking for repaired proofs (Section 3.3).

3.1 The Task: Proof Repair

In machine learning, a *task* refers to a high-level input/output specification of what is being learned. A dataset and benchmark suite typically organizes itself around a particular task while remaining agnostic to the details of the model implementation.

We define proof repair as a machine learning task as follows:

1. **Inputs**: old theorem, proof of old theorem, and new theorem
2. **Outputs**: proof of new theorem

Note that this particular proof repair task assumes that we already know how to repair the theorem statement and all of its dependencies. We could also consider a second proof repair task that allows the model to also repair the specification itself. This second proof repair task would be harder to evaluate, so we do not focus our benchmark suite on it at this time, though our dataset supports this. We discuss our plans for measuring success on the second task more in Section 3.3.

Inputs

TODO: Show an Example. We aim to provide sufficient context in the data to support a wide range of machine-learning approaches and repair model architectures. At a high level, the input to the machine-learning repair model is the entire state of a project where the

[1] We plan to release the dataset before the camera ready. The research team includes government contractors who are subject to US government approval processes, and cannot release datasets and code without a month's notice. We have included a small sample as supplementary material in our submission, which we were able to approve a month in advance.

approach dictates how much of this state (and in what form) actually reaches the model. More precisely, we expect the input to comprise the statement of the theorem whose proof should be repaired, any contextual definitions on which it depends, the step-by-step goals and hypotheses for each sentence in the old proof, and known changes to the project up to and including changes in the theorem statement and its dependencies. For each of these components, we supply raw text representations (as one would observe in source code or in feedback messages within *CoqIDE*), abstract syntax trees (ASTs), and identifiers from which one could hypothetically compute data flow graphs. In the case of goals and hypotheses, serialized Coq kernel representations supply detailed internal proof states. In addition, environmental dependencies such as Coq compiler version are captured for errors induced by external application programming interface (API) changes.

Outputs

TODO: Show an Example. The ultimate output is the repaired proof, which takes the form of text that may be generated one sentence at a time, all at once, or through targeted modifications of existing sentences. In the case of supervised repair learning, we supply ground truth targets in the same form as the inputs: raw text, ASTs, etc. for the entire repaired project's state (compactly represented as a 'diff' relative to the input). Sufficient context is provided in the data to programmatically execute up to the error in an interactive REPL such as *coqtop* or *sertop* , where one may apply reinforcement learning akin to CoqGym.

3.2 The Data: Aligned Git Commits

The training and testing data comprise of aligned Git commits for a selection of realistic Coq projects. In total, it includes n commits for m Coq projects, with p repair examples in total. TODO: Summary of other important stats goes here. A full summary of all projects and commits in the dataset is in Table 6 in Appendix A.

Choice of Coq Projects

Projects were originally selected by querying GitHub's API for projects that

1. contained Coq source code,
2. had a file called 'Makefile' in the project's root directory, and
3. had at least 100 commits to mine repair data from.

Eventually, we included CoqGym [51] and filtered to projects that were listed in opam repositories. Some projects were excluded for not containing any proofs or for having ulterior motives in their builds (e.g., projects that intended to test the performance of the Coq compiler *coqc* rather than build fascinating proofs).

TODO: How do we know these are reasonable Coq projects to use for this task? Also note we are trying to broaden beyond what we currently have, but infra challenges, tease to challenges section, discuss later.

TODO: Enforceable note about right to delete

Repair Examples

Within each Coq project, the data comprises a number of repair examples—that is, changes to definitions or proofs. A repair example is constructed by comparing a definition or its proof before and after a change. Since sentences, identifiers, and files may be moved, renamed, added, or deleted between commits, they must first be *aligned* to ensure the right changes are compared. At a high level, this means that Vernacular commands in one commit are assigned one-by-one to commands in another commit, where these assignments may cross file boundaries. Note that each command may not get a partner, indicating that it was either deleted or added. We describe this in more detail in Section 4.

After alignment, proof repair examples are constructed by partially applying changes, e.g., by omitting the changes to a proof that accompanied a change to the proposition. Thus, one pair of commits may give rise to multiple repair examples. The repair examples are compactly represented simply by commit hashes and diffs that indicate the state before and after a repair. This compact representation enables dissemination of the dataset without the accompanying projects cache, although we note supplementary tools for efficiently extracting project data will still be vitally important for eliminating redundant computations and effort in practice.

Data Split

Machine-learning datasets and benchmark suites often include a data split between training data, validation data, and testing data. We do not commit to a single split ahead of time, but we consider two different ways of splitting data:

1. across projects, and
2. chronologically within projects.

These two splits test two different kinds of generalization beyond the training data, so we may consider them two different benchmarks for the same proof repair task. We find it reasonable to consider both of these kinds of generalization in measuring success of a proof repair model, so we plan to include defaults for both splits in the final release of the dataset and benchmark suite.

Across Projects

The first split—across projects—chooses distinct sets of Coq projects to use for training, validation, and test data. This is standard in existing benchmark suites for proof generation like CoqGym [51], and effectively measures generalization of the learned model to new projects not seen at training time.

Chronologically

The second split—chronologically within projects—uses the same set of projects for training and test data. However, these two sets in particular are split chronologically, so that training data includes earlier commits, and testing data includes later commits for the same projects. This effectively measures generalization of the learned model to new changes within a given project, when the model was trained on older data for that project.

3.3 The Metrics: Proof Checking

Changes in proof developments that break proofs can be fixed in two ways: by repairing the proofs themselves, or by repairing some other definition, like a program or specification [38]. Our dataset includes both kinds of changes. For the sake of metrics for evaluation, we focus our benchmark suite on the former (repairing proofs), as the metric for success is immediately clear. We hope our benchmark suite will also be useful for the latter (repairing definitions), but we believe the problem of choosing a good metric for success for repairing definitions to be an open research problem.

Repairing Proofs

We focus our initial benchmarks on the problem of repairing a proof script assuming that the statement of the repaired theorem to be proven is already known. In this case, checking the correctness of the repaired proof amounts to using Coq's kernel to proof check the type of the repaired proof against the type that represents the desired repaired theorem statement.

The metric of proof checking is the same as that used for the standard proof generation benchmark suite for Coq—CoqGym [51]. This metric is sound and complete (up to the correctness of Coq's trusted kernel, and designating nonterminating proof scripts as incorrect proof scripts), as:

1. any proof that checks with the desired type is a proof of the theorem the type encodes (**soundness**), and
2. all proofs that prove that theorem will check with the desired type (**completeness**).

In this sense, for this flavor of proof repair, we are able to take advantage of the fact that proof checking is a perfect oracle when the theorem statement is known. The presence of a perfect oracle has been hugely beneficial for existing machine learning work for proof generation [21], and for early symbolic work on proof repair when specifications do not change [42]. It continues to benefit machine learning for proof repair.

Repairing Definitions

The metric of proof checking is useful when the repaired specification and all of its dependencies are known in advance. But what when they are not? Helping users fix the definitions that the specification itself depends on—or the theorem statement itself—is also highly desirable. In fact, a recent user study of eight intermediate through expert proof engineers in Coq found that 75% of the time those proof engineers fixed a broken proof throughout the course of the user study, they did so by fixing something else, like a program or specification [41]. So there is reason to suspect that supporting this use case may actually be *more* helpful to proof engineers than supporting the original flavor of proof repair.

We believe the problem of finding a suitable metric for this use case to be an open research problem. Existing metrics are insufficient:

- The metric of **proof checking** is no longer sufficient when the repaired specification is not fully known in advance, as showing that the proof type checks is not meaningful unless we know the type it ought to have.
- The conservative metric of **exact equality** with an expected repaired specification or definition is sound, but far from complete, as there are many equivalent ways to state the same theorems or write the same definitions.
- Loosenings of exact equality to use, for example, common notions of **definitional equality** or **propositional equality** in Coq are slightly less conservative, but are still far from complete.
- Proof repair across type equivalences [40] defines a more complete notion of preserving thespecification correctly across a change in datatype in terms of **univalent transport**, but this notion is too narrow to express some changes in specifications that add or remove behaviors or information. Furthermore, checking this metric automatically in a benchmark would be difficult, as it is undecidable without a generated proof of correctness of the change, and requiring proofs of correctness of the change would place a very large burden on repair tools.

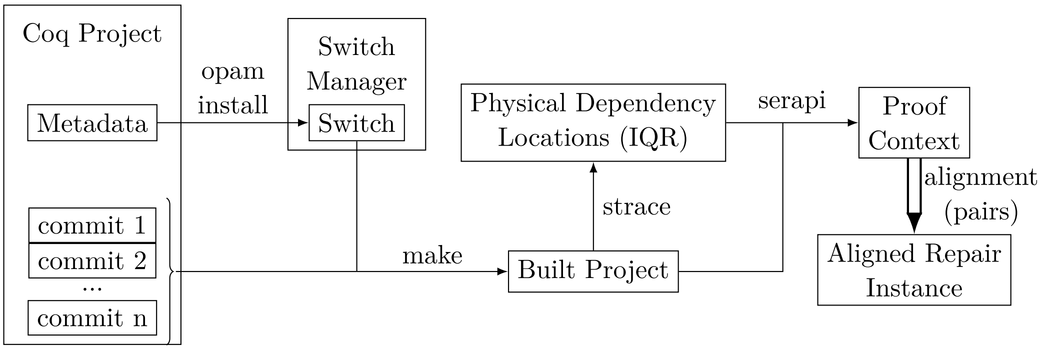

Figure 2 Process of extraction for a Coq project commit.

- In other domains, it is common to use **distance metrics**, but common distance metrics like BLEU [31] are poor measures of success in program synthesis tasks TODO: cite. TODO: (AG) There is also Code-BLEU, but its computation is complicated by the difficulty in obtaining ASTs and data flows As formal proof developments are in some sense programs, we expect existing distance metrics like BLEU score to be inadequate for proofs as well. Distance metrics do have the distinct advantage of being continuous in some sense—they make it possible to define what it means to be "close to correct." But being "close to correct" does not mean much when the notion of correctness used is itself incorrect.

For now, we largely focus on the proof repair task where the repaired specification and all of its dependencies are known in advance, as it is easy to measure success on this task. Still, the data for the task of repairing definitions like programs and specifications is present in the dataset. If you choose to train and evaluate a model for the task of repairing definitions, we recommend using a conservative metric like exact or definitional equality, so as to avoid the danger of chasing misleading benchmarks alluded to in Section 2. We hope the community will come together to develop a suitable less conservative metric going forward.

4 Building the Proof Repair Dataset

Now that we have introduced the proof repair dataset, we take a step back and describe the processes behind our data collection efforts. The process of generating repair data from a project is summarized in Figure 2 and comprises the following steps:

- We start with a collection of aggregated open-source **Coq Projects**, which form the foundation of our dataset (Section 4.1).
- We determine how to build each project using the **Switch Manager (SwiM)**, which takes the dependencies of a Coq project and gives a copy of the cached *switch* (OCaml Package Manager (opam) sandbox or virtual environment) that satisfies as many of them as possible and in which the remainder can be installed (Section 4.2).
- Once we have a switch, we run the build command in the generated switch to produce a **Built Project** (Section 4.3).
- We `strace` the build process to scrape the **Physical Dependency Locations (IQR flags)** of each document in the project, which lets us know the mapping between Coq imports and physical disk locations (Section 4.4).

- Using the IQR flags for each document in a built project along with the Coq serializer SerAPI [8], we extract a **Proof Context** corresponding to each intermediate proof state for the project by querying Coq's state during the execution of proofs (Section 4.5).
- Finally, we align changed proofs across commits and save those along with their intermediate proof contexts to arrive at an **Aligned Repair Instance**, which is a final product in our dataset (Section 4.6).

Building this infrastructure was a significant undertaking with many challenges encountered along the way; we discuss these challenges in Section 5. Our hope is that the infrastructure we have built will make it easier to collect similar datasets in the future.

4.1 Coq Projects

The foundation of the dataset comprises open-source Coq projects found in publicly available repositories on web hosting services (primarily Github). The choice of projects is summarized in Table 6 of Appendix A. Mining the commits of these projects eventually yields examples of refactors or repairs. Each project is also accompanied by per-commit metadata containing project dependencies, source URL, and build commands that is in parts manually curated and programmatically inferred.

4.2 Switch Manager

To extract information about these projects like intermediate proof states, we need to be able to build them. This requirement is nontrivial because different projects and even different commits of the same project can include different versions of dependencies—including different versions of Coq or of the OCaml compiler.

To resolve dependencies and make it possible to build many different commits of the same project, we introduce a novel SwiM capability that works in tandem with opam—an OCaml package manager that acts as the primary distributor of Coq projects—to extract a Python object that models the build environment for a given commit of a given project. The Python object in particular models an opam switch, which is the way opam represents dependencies along with OCaml compiler versions.

This capability subverts the typical opam workflow that requires one to manually create a switch and to set environment variables for the current shell to activate that switch. This manual workflow would be intractable at the scale of hundreds of commits for each of hundreds of projects. With the SwiM, we can automate this functionality and extract a dataset at scale.

A number of benefits arise from the design of the SwiM. First, the SwiM enables sandboxing of project builds by providing temporary isolated switch clones that last for the duration of the commit's extraction. Furthermore, the SwiM minimizes the time required to obtain a clone by maintaining a pool of switches with pre-installed packages and choosing the one upon request that is closest to satisfying a commit's requirements.

As new commits are built, switches containing their dependencies are added to the managed pool of switches. Since switches range in size from hundreds of megabytes to a few gigabytes, a least-recently-used cache maintains the total disk consumption below an implicit limit by deleting stale switches that are not frequently used. A shared switch manager facilitates multiprocessing with a shared pool of switches by queuing requests from concurrent threads/processes. Implementation of this capability required reflection of opam's dependency formula parsing and evaluation logic from OCaml to Python.

4.3 Built Project

Using the switch provided by the SwiM and a build command from the metadata, we may be able to build the project. Confounding issues that may prevent building include undefined opam variables within dependency formulas, for which we only provide default values of "True" for each of the `build`, `post`, and `dev` builtin variables. In practice, we have so far seen a build failure rate of about 68%. We attempt to build each commit with seven different major versions of Coq ranging from 8.9 to 8.15 corresponding to versions of SerAPI that support capabilities deemed necessary for data extraction. Since Coq releases are rarely backwards compatible, many of the build failures can be explained by the fact that each commit can only be expected to build without error for the single Coq version for which it was written. Furthermore, since we pin one of the seven Coq versions in the switch supplied by the SwiM, conflicting version requirements may yield an opam command that has no solution. Consequently, some build errors are inevitable.

However, other errors are due to mistakes or missing information in the human-sourced metadata. We plan to address this latter class of build errors over time by fixing problems in the metadata through automated inference mechanisms. If the project build fails, we hope in the future to be able to recover proofs from the documents that built before the failure as well as subsequent independent proofs. We are also exploring ways to automatically recover from simple build errors such as dependency mismatches between the switch and the project's requirements by using the date the commit was made as a version hint.

4.4 Physical Dependency Location (IQR)

In order to run any of the Coq or SerAPI tools (e.g., `coqc`, `coqtop`, `sertop`) on a given Coq source file, many Coq source files require one or more flags to be passed to these commands to specify the physical location of dependencies. These flags are described below:

- The `-I` flag allows a directory to be added to the OCaml loadpath
- The `-Q` flag adds a physical directory to the loadpath and binds it to a specified logical path
- The `-R` flag does the same as the `-Q` flag, except that subdirectories are also made available recursively

In publicly available Coq projects, these flags (informally referred to as "IQR" flags from here on) are specified in one or more build or configuration files. There are several common approaches for specifying IQR flags, but there is no single standardized approach, making it difficult to automatically infer these flags from configuration and build files alone. While projects will be able to build successfully without our inferring of these flags, we need to infer them in order to use SerAPI tools in other stages of our framework.

Our solution to this problem builds off an approach developed in IBM's PyCoq [2]. Following PyCoq, we use `strace` to inspect the actual commands run during the build process for a Coq project. Each build command is captured and any present IQR flags are extracted using regular expressions. In some projects, build files may be nested, and IQR flags may specify physical paths that are relative to the nested directories. We need to ensure that the inferred IQR flags are relative to the project root directory, so before we store the inferred IQR flags, their paths are resolved to the project root directory.

[2] https://github.com/IBM/pycoq

4.5 Proof Context

Once the project has been built, the individual Coq source files are parsed into sentences and then interactively executed with `sertop` to capture intermediate proof states. A parser (`sercomp`) is available through SerAPI, but it can only be used on Coq source files whose dependencies have already been compiled, which prohibits its use in planned recoveries from partial builds. Furthermore, `sercomp` introduces significant overhead through computation that will be redundant with `sertop`.

From this, we can collect thorough context from the document, such as whitespace-normalized text, ASTs, command types, and intermediate proof steps with goals and hypotheses. Each command is accompanied by inferred identifiers of the command itself (e.g., the name of an inductive type and its constructors) and a list of fully-qualified identifiers referenced within the command, which enables models to more easily incorporate relevant local context or apply graph-based approaches. Any errors that are encountered in the execution of a command are cached as well for repair models that can adapt their approach based on the error. Accompanying source code locations allow for accurate provenance of data and application of proposed repairs to appropriate destinations for testing.

4.6 Aligned Repair Instance

The final step in our data collection process is extracting proof repair examples from different versions of the projects. To extract these examples, we need a way to tell how different versions of definitions and proofs across different commits correspond to each other.

We approach this as an alignment problem. Because lines in one commit are not necessarily located at the same index as in the other commit, we first have to establish a robust mapping between lines that can handle the changes usually associated with a commit: lines can be changed, deleted, added, and rearranged. This is similar to the 'diff' utility, which essentially tries to explain what changes were made between two files, but it lacks capacities we deem necessary, such as matching lines that were rearranged and matching lines that were only changed slightly. In particular, we wish to establish a 'diff' in terms of Coq Vernacular commands treated as atomic entities that can capture whether an individual definition, lemma, inductive or other identifiable entity was moved, renamed, or otherwise altered during a refactor.

A traditional order-preserving alignment between two sequences TODO: e.g., CITE is not quite an appropriate approach to resolve this issue as it cannot correctly align two independent definitions whose order has been reversed during a refactor (perhaps due to an introduced dependency). Instead, we must consider permutations of commands within and across file boundaries. We approach the problem as a bipartite matching or *assignment* between the elements of two sets such that the overall similarity of matched elements is maximized. We can formally specify the desired assignment between two commits X and Y considered as respective sets of m and n commands across one or more files as the solution to the following optimization problem:

$$\begin{aligned}
\underset{f}{\text{minimize}} \quad & \sum_{x \in X} \sum_{y \in Y} C(x,y) f(x,y) \\
\text{subject to} \quad & \sum_{y \in Y} f(x,y), \leq 1, \\
& \sum_{x \in X} f(x,y) \leq 1, \\
& \sum_{x \in X} \sum_{y \in Y} f(x,y) = \min\{m,n\}, \\
& f(x,y) \in \{0,1\}
\end{aligned} \tag{1}$$

Here, $C(x, y)$ is a non-negative cost function that measures the *distance* between the commands x and y, and $f(x, y)$ indicates the binary assignment of x to y. The constraints, in order, state that no element of X can be assigned to more than one command of Y, no element of Y can be assigned to more than one command of X, and as many commands as possible must be assigned (i.e., choosing to assign nothing in an attempt to minimize the cost is not allowed). This optimization is an instance of the well-known *assignment problem*, which one can solve exactly in polynomial time according to the Hungarian algorithm [30] (or one of its many alternatives) or approximately in linear time according to the Sinkhorn algorithm [17] (which also admits GPU acceleration).

The optimization is parameterized by the choice of C, for which we choose a normalized variant of the Levenshtein edit distance E:

$$C(x,y) = \frac{2E(x,y)}{\|x\| + \|y\| + E(x,y)}, \tag{2}$$

where $\|x\|$ and $\|y\|$ give the character lengths of x and y considered as text (not including proof bodies). This normalization is an instance of the biotope or Steinhaus transform [18], which preserves the metric properties (such as the triangle inequality) of the edit distance. We further threshold the distance by a constant t such that $C_t(x,t) = \min\{C(x,y), t\}$, which also preserves metric properties [32]. After solving for f, commands x and y assigned to one another ($f(x, y) = 1$) with a cost of t are considered to be unassigned (i.e., we determine that x was dropped between commits and y was added). We choose $t = 0.4$, which roughly corresponds to 50% of a command's text being changed before it is considered to have been dropped. TODO: (AG): This is a point for discussion as we could also introduce some alternative heuristics for checking whether two assigned commands are the "same".

Solving this assignment problem for two entire commits can be costly: solving for the assignment exactly is cubic complexity, and the edit distance must be calculated for all pairs of commands from both commits, which is necessarily quadratic complexity. Furthermore, the assignments produced may be somewhat spurious, especially in the event of multiple global optima. Though we cannot enforce preservation of order as an absolute constraint, we do wish to apply it as a pseudo-constraint to resolve ambiguity. We mitigate these issues by applying the assignment problem only to those commands known to have changed in some manner between the commits according to their intersection with a (Git) 'diff'. The final resolution of the problem is thus somewhere in between alignment and assignment.

Once an alignment is determined, we create examples of proof errors by leaving out changes to individual proofs one at a time, thus providing the context for each change to a proof that required repair but not the repair itself. The left-out change to the proof then accompanies the error as a ground truth target for supervised learning.

5 Challenges

Machine learning for proofs will likely continue to flourish as a field whether or not the core proof assistant community participates. But if we do participate, perhaps we can work to ensure that it grows in the right directions—towards the tasks and benchmarks that matter, equitably across proof assistants.

We have a long way to go before that is true. Collecting and building datasets and benchmark suites for many tasks is still extremely challenging, and it is challenging in a way that is not at all equitable across proof assistants.

Here, we discuss challenges we faced in building this dataset and benchmark suite that fall into three categories:

1. Archival & Versioning (Section 5.1),
2. Package Management & Distribution (Section 5.2), and
3. Parsing & Serialization (Section 5.3).

For each of these categories, we discuss our experiences dealing with those challenges, what we believe the Coq community can learn from other proof assistant communities, and how the proof assistant community at large could address them more sustainably going forward.

5.1 Archival & Versioning

One of the largest barriers to building this dataset was the lack of a centralized archive for Coq proof data, combined with the difficulties of dealing with different versions of different projects and their dependencies. These interacted in complex ways, the end result of which was a scattered source of data that was difficult to aggregate in a way that would be useful for downstream machine learning tools, especially for the proof repair task.

5.1.1 Our Experiences

We faced four challenges related to archival and versioning:

1. the **lack of centralized archival** of Coq proof developments,
2. frequent changes in **proof assistant versioning** that broke the abstraction barrier,
3. significant **build system variation** across different proof developments, and
4. tracking information across **changing definitions & proofs** within proof developments.

Lack of Centralized Archival

Coq lacks a centralized archive that it is standard for proof engineers to use. There are a few archives that have existed at various points in time, but there is not a strong norm of using them. Consequentially, we had to aggregate proof developments from public sources like GitHub. While this worked nicely, it also meant that proof developments lacked useful associated metadata, or were formatted in different ways, making automatic aggregation, parsing, and building more challenging. This perhaps is at the root of many other challenges we faced and had to overcome.

Proof Assistant Versioning

One of the barriers to centralized archival in Coq is the large variation in proof assistant versioning, while the up-to-date archive that does exist requires proof developments to be on the latest version. There was also notably not a uniform way to denote which proof assistant version a given proof development was on, nor was there a uniform way to interact with Coq across proof assistant versions. The tools we used for serialization and parsing, for example, were highly version-dependent (see Section 5.3). This meant that we had to infer proof assistant versions using imperfect proxies, and build data collection tools that inherently depended on those inferred versions.

Build System Variation

Over the years, the recommended build system for Coq proof developments has been in flux. In 2019, for example, the Coq development team urged proof engineers to move their proof developments to Dune[3]. This did not fully succeed, and the documentation for the latest Coq version includes instructions for both Dune and the native Coq build system[4]. The native Coq build system itself has also changed over time in significant ways, losing compatibility with previous versions of the same build system. For example, the arguments to the `coq_makefile` command changed over time. Because of this fragmented build infrastructure, we had to employ extremely abstract methods to extract information whenever a build system was involved, such as using `strace` to grab flags passed to Coq's compilation command `coqc` during builds (described in Section 4.4), while making almost no assumptions about the process invoking `coqc`.

Changing Definitions & Proofs

Building a proof repair dataset meant mining not static proof developments, but rather changes in proof developments over time. Figuring out which definitions and proofs corresponded to one another across versions was a significant challenge. This challenge is one that the REPLica user study also faced; rather than custom alignment, they relied on GitHub's diff command, but needed a human to manually crawl the result to find repair examples [41]. Since our data was less granular than REPLica's, we were able to make the simplifying assumption that all repair examples occurred commits that occurred immediately after one another, without intermediate commits. We then mined repair examples automatically using the alignment algorithm discussed in Section 4.6.

5.1.2 Other Proof Assistants

Other proof assistants can learn from Isabelle/HOL's rich archival culture. In Isabelle/HOL, it is standard to upload proof developments to the Archive of Formal Proofs (AFP). Isabelle/HOL's AFP is highly centralized, and also makes it easy to associate proof developments with metadata that may be useful for machine learning. It is also neatly versioned, with all proof developments updated for every minor and major version of Isabelle/HOL, and with major version semantically grouped in different folders.

At the time of writing, the AFP includes 725 proof developments[5]. The AFP already forms the basis of a dataset for Isabelle/HOL for static data [23]. We suspect it will also make for a very strong basis for a proof repair dataset due to its neat versioning. We believe that other proof assistants should take influence from Isabelle/HOL in developing a large centralized archive that is standard to use and neatly versioned, and that makes it easy to associate proof developments with metadata useful for machine learning tasks.

---

[3] `https://coq.discourse.group/t/proposal-a-custom-build-tool-for-coq-projects/239/2`
[4] `https://coq.github.io/doc/master/refman/practical-tools/utilities.html`
[5] `https://www.isa-afp.org/statistics/`

5.1.3 Recommendations

There are still a number of ways that even the best archival and versioning infrastructures for proof assistants fall short. For example, while Isabelle/HOL's AFP makes a natural data source for machine learning tools for proofs, it does not include any processes for informed consent—and the datasets that build on it assume that all publicly available data is fair game. While this is standard in machine learning for code and proofs, it is not ideal. Consent should be considered at every step in the process, and archival is a natural place to consider that.

Another problem with archives is keeping them up-to-date. Is this sustainable as archives continue to grow, while the cost of repair is still so high? We are not sure how the AFP maintainers manage this burden, but we assume that not all communities can manage it, and lowering the cost of repair will be central to better archival across all proof assistants, and to getting proof developments on the latest version to archive to begin with.

Finally, while archival makes it possible to associate metadata with proof developments, there is little consideration for adding metadata that associates definitions and proofs *across versions* of a proof development. This kind of metadata would make building repair datasets and benchmark suites much easier, and make it simpler to address the problems of change as they interact with archival in the future.

Based on our observations, we make the following recommendations for the proof assistant community going forward:

1. Work with the Isabelle/HOL community to learn how to build **centralized archives** as successful as the AFP for other proof assistants.
2. Include an **informed consent form** in any centralized archive that allows proof engineers to opt in or out of their developments being used for machine learning tools.
3. Create **better and more consistent standards** for tools like build systems.
4. Help proof engineers **port legacy proof developments** to meet those standards.
5. Continue to work on tools for **proof repair and reuse** that ease the burden of porting said legacy developments.
6. Determine what **kinds of metadata** both within and across proof developments would ease the development of machine learning for proofs tools for the tasks that matter most.
7. Make it easy to **track that metadata** inside of any centralized archive.
8. Create standard ways of **associating that metadata** with proof developments even outside of centralized archives.

5.2 Package Management & Distribution

The beating heart of our collection effort is package management, which gives us a programmatic interface to building compatible environments for the data-set's constituent projects. In our case, we use opam: a popular OCaml package manager and the primary distributor for Coq packages due to the close relationship between the OCaml and Coq communities. TODO: OPAM isn't supposed to be in caps, even though it's an acronym. The opam package manager was nominally designed to service individual developers using a few switches, not use-cases like our own where we have to spin up dozens of switches efficiently. This required us to reimplement and expand upon some of opam's capabilities.

5.2.1 Our Experiences

We faced three challenges related to package management & distribution:

1. Packages in opam have very **expressive dependencies** which complicate efficient installs.
2. The opam package manager has no mechanism to cache builds, so we need to **copy switches** to avoid rebuilding the same packages.
3. It is difficult to use opam with packages that have **inconsistent dependencies**, though this is desirable to us.

Expressive Dependencies

The opam package manager provides a powerful and expressive syntax (package formulae) for packages to specify dependencies over other packages. Package formulae allow package developers to restrict the versions of dependencies that can be installed, to conjunct and disjunct formulae into more complicated expressions, and to refer to variables declared elsewhere in the environment. This benefits the library developer, who can precisely specify the environment that their code will run in, but for our purposes package formulae pose a challenge: packages can be very picky about their environments and force opam to rebuild existing libraries. Since we need to install many versions of many packages, we needed to look into efficient ways to create and select switch environments they could be installed in, which meant interpreting these expressions. As a result, we had to reimplement a majority of opam's package formula features, including parsing the custom grammar for package formulae and implementing package version comparison, in order to reason about what existing switches would be the install environment for a package where opam would do the least amount of work to install the new package.

Copying Switches

If multiple opam packages conflict with each other, they can be installed in separate switches. Since we were installing packages from various points in the past, packages frequently required different version of dependencies, so they frequently conflicted. This meant we had to spin up many switches, but spinning up a switch required us to rebuild packages from source each time. We reasoned it would be better to build a package once and deploy it in multiple switches. However, many executables built by opam contain their absolute path as a hardcoded variable, which means these executables stop working if the name or location of the switch changes. In other words, a compiled opam package only necessarily works in one switch.

Our workaround was to copy switch directories without informing opam, which allowed us to start a new switch using another one as a base, saving valuable build time. Whenever a copied switch needed to be used, we bind-mounted the copied switch over the original switches location, so a copied switch effectively shared the same path as its parent. We used the `bwrap` utility to allow unprivileged processes to bind-mount, which is already used internally within opam to sandbox package builds. Of course, handling these cloned switches required additional bookkeeping and infrastructure, complicating our efforts.

Inconsistent Dependencies

For our repair dataset, we may like to see how a package breaks when dependencies are updated, since this breakage is another repair instance. Naturally, opam does not want to let us install packages it knows are incompatible. It is possible to force opam to install packages with inconsistent dependencies anyways through the use of the `ignore-dependencies-on` flag, but this puts the solver in an inconsistent state. Trying to use opam after inconsistent packages are installed typically results in opam trying to change the versions of packages as soon as it can, even without being prompted to do so. Attempting to pin both packages to inconsistent versions renders opam inconsolable, refusing to do anything with dependencies until the inconsistency is resolved. Finally, it is possible to continue using opam as long as you use the previously stated `ignore-dependencies-on` flag once again to disregard all dependencies on the wrong-versioned-package, but if you install another package that depends on the package whose dependencies are being ignored, it will simply install the newest version, not necessarily one that is compatible (ignoring all dependencies, not just the one exception created initially).

A cleaner way to tamper with dependencies is to edit the opam package file of the package of interest directly so that opam's dependency solver isn't involved in the process. opam even provides a mechanism to pin local, modified versions of packages and a command line tool (`opam pin edit`) to open a package's metadata in an editor, which are useful for package tinkering and allow inconsistent packages to be solved without causing the above issues. Unfortunately, this is only a complete solution for humans editing a few packages. An automatic tool, such as would be useful for our efforts, still has to parse the package metadata fields, parse the dependencies, modify them, and paste them back in the right spot.

5.2.2 Other Proof Assistants

Agda has its own library management system, which it uses in combination with hackage (to install Agda itself). Anecdotally, researchers we have spoken to cite installation difficulties as a barrier for picking up Agda. Lean similarly has its own package manager, but lacks advanced features for the kinds of problems we faced. Isabelle/HOL's AFP serves as a de facto group of packages to install. But Isabelle/HOL in general takes a very IDE-centric approach to builds and other tooling [39, 47] one of the authors has found that students learning Isabelle/HOL in a proof automation course struggle to understand how to build dependencies.

In summary, we are not aware of an elegant and effective solution to package management for other proof assistants. We are eager to hear from experts in other proof assistant communities about whether the outlook is any better in other proof assistants, and about what we can learn from any solutions that do exist. If the situation is in fact universally bleak, perhaps we ought to come together to innovate.

TODO: It's also possible to use nix for Coq, and this uses online caches so you don't need to rebuild. Why don't people use this? Not sure where to drop this but it's an important note I think. Also, Isabelle sessions are a thing, and can inherit from other sessions, solving some but not all of the caching issue. Also the `-R` option. See Tweet responses and consider.

5.2.3 Recommendations

There is a legitimate argument to be made that package management by default targets a very different use case from ours, and perhaps existing tools are sufficient for that use case. If that is the case, it may make more sense to build shared high-level libraries and tools on top of existing package managers to support bulk cases like our own, especially since these use cases are common when building machine learning datasets and tools for formal proof.

Nonetheless, there may be room for improvement of the package managers themselves. For example, the problem we encountered of copying switches was due to poor caching of build dependencies, which itself was due to some degree to hard-coding of paths. It may be better if applications could consult something more malleable than a compile time string to determine if the applications consulted something more malleable than a compile time string to determine these paths, like an environment variable, which opam already configures several of over the course of normal switch operation. This would likely require effort on the part of each of the applications, but may help with the problems we encountered and more.

In all, we make the following recommendations for the proof assistant community going forward:

1. Consider building **shared libraries and tools** optimized for the problems of bulk builds.
2. Consider limiting the scope of possible **dependency complexity**, or else building better tools to resolve complex dependencies efficiently.
3. Work on better **build caching** across multiple or bulk builds, for example by avoiding the hard-coded paths seen in opam.
4. Provide more effective ways of **overriding conflict declarations** that are overly conservative.
5. Work together across proof assistant communities to determine **common lessons and infrastructure** to resolve these problems.
6. Consider opening conversations with **language developers and companies** outside of verification about their package management and distribution solutions, as this problem is pervasive across all software.

5.3 Parsing & Serialization

TODO: Radiance folks and Tom: I need help with 5.3.0 and 5.3.1 before it's in a state such that I can write a good 5.3.2 and 5.3.3. I'd appreciate if you all can take a good solid pass at this.

TODO: Intro paragraph that is higher-level than what we had before. No matter how sophisticated the build system, we cannot get detailed data about individual proofs without parsing the Coq files and serializing proof state to text. SerAPI [8] is the de facto standard for serializing interactive Coq data, providing a query protocol for exposing aspects of internal Coq program data including definitions in the global environment, syntax trees, goals, types, and more. We used the CoqGym [51] Python wrapper as a starting point for our implementation, taking care to decouple it from CoqGym's custom versions of Coq and SerAPI since we need to support multiple versions of each coinciding with chosen projects' Git histories. This need to support multiple versions of Coq exacerbated challenges arising from gaps in SerAPI's query protocol, requiring us to implement workarounds using the most public and arguably stable interface Coq possesses: its Vernacular query commands.

5.3.1 Our Experiences

We faced five challenges related to parsing & serialization:

1. Executing a file one Coq sentence at a time requires accurately **parsing sentence boundaries**, but parsing requires execution: a catch-22.
2. **Calculating command dependencies** requires identification of novel definitions and full qualification of embedded references with library prefixes, neither of which are provided through SerAPI queries.
3. **Determining the scope of a conjecture** is complicated by the potential presence of nested proofs/definitions, arbitrary grammar extensions, and the lack of obligatory syntax to delineate proof boundaries.

4. Some proof developments consume an enormous amount of memory and processor time, requiring **resource limits** to keep data collection tractable.
5. SerAPI is experimental software, which leads to breaking **changes between versions**.

TODO: Add challenge for error handling and recovery with discussion of regression proving and asynchronous proof checking and why despite its availability as a feature of Coq, it is not an option for us (would require invasive modification of build processes for every commit, deemed impractical; also does not work for all projects). Also, not clear how asynchronous proof checking would help with extraction of intermediate proof states except that it implies there is a way to determine conjecture scopes without evaluating them (actually, the way would be to use the Vernacular classification functions alluded to later).

Parsing Sentence Boundaries

A Coq statement or 'sentence' ends with a period (.), but Coq also uses the symbol for import paths and module members so that one cannot identify sentences in a file merely by splitting on periods. To further complicate matters, Coq boasts an extensible syntax that can greatly improve the read and writability of Coq expressions, but allows a user to define syntax that allows periods to show up in even more places.

An example of a problematic extension of '.' is shown in 3. This snippet defines syntax to create tuples using a period rather than a comma, which complicates sentence splitting to the point where the latest version of *CoqIDE*, the official editor for Coq, cannot correctly parse and run this code even though Coq itself can.

```
Notation "( a . b )" := (a, b).
Check (1 . 2).
```

Figure 3 This example was found (here).

We did not discover any public or officially supported mechanism to extract the sentences of a Coq document. Though one could hypothetically parse the output of *coqdoc*'s HTML or LaTeX output, correlating the sentences back to their original source code locations (line, column, and character numbers) brings additional complexity. We opted to implement our own heuristic regular-expression-based parser in Python for simplicity and maximal portability between build environments, noting that it inherits weaknesses exhibited by *CoqIDE* but reasoning them equally acceptable as the official editor. A custom string subclass tracks locations of individual characters across splits, slices, and concatenations of the parsed source code.

Calculating Command Dependencies

To identify the dependent definitions referenced in a statement, we must associate with each applicable command the identifier(s) that it defines and references. No SerAPI query provides the name(s) introduced by a given command, nor is there any unambiguous syntax one can use to identify a name that will be visible in the global environment that also generalizes across builtin Vernacular commands (and Coq versions) or unforeseen grammar extensions. Furthermore, identifiers within ASTs yielded from SerAPI typically match their qualification in the source code and thus are not guaranteed to unambiguously identify referenced or shadowed definitions when examined out of context.

Instead, we rely upon parsing user-level feedback and queries for detecting definitions. Though Coq typically emits feedback of the form "X is defined" when a new function or inductive type is introduced, it is not guaranteed to do so for a proposition when its proof is completed (whether it does or not depends upon the proposition type and Coq version). Consequently, a Vernacular `Print All` command is inserted when no such feedback is detected to print the names and types of all definitions introduced since the start of the interactive session. Monitoring changes in the set of all locally defined names then allows one to deduce which definition, if any, was introduced by a given command.

We qualify identifiers in an AST through SerAPI's `Locate` query, which requires a separate query for each identifier. These repetitious queres add a significant amount of unavoidable overhead. Care must be taken to ensure locally bound variables within binders, patterns, or other sub-expressions do not get incorrectly qualified as any top-level definition that they may shadow. Our solution is ultimately limited by the lack of accurate scope rules that would require access to or replication of Coq's internal name resolution capabilities. Aside from being unable to disambiguate locally bound variables sharing the same name, we also note one restriction on resolving globally bound identifiers: if a new identifier shadows an existing one, then the defining command cannot reference the shadowed identifier. Violation of this assumption is possible (consider a recursive function `nat` that expects arguments of type `nat`) but not expected to be a significant risk as it is unlikely in the first place and would generally be considered poor practice. If the restriction is violated, then the shadowed identifier will simply be shadowed starting with the violating statement.

Note that all of the above solutions require parsing identifiers to some degree, which poses its own challenge due to Coq's versatile support for Unicode. The character set comprising Coq identifiers intersects each of the usual meta characters used to denote whitespace (non-breaking space 0x00A0), word characters, and other symbols. The exact character set is not publicly documented. Consequently, one cannot accurately construct a regular expression for parsing Coq identifiers without duplicating the Unicode tables and implied regular expressions within Coq's source code, which is what we ultimately resorted to as we could not eliminate false positive and false negative detection of identifiers with handcrafted regular expressions.

Determining Conjecture Scope

Determination of conjecture scope decomposes into two subchallenges: attribution of proof steps to the correct conjecture and detection of proof (conjecture) completion. Though one can query the open goals at any given time, the conjecture name cannot be derived from the information (and identical goals can appear in different proofs). Furthermore, steps of distinct proofs may be intermingled or nested as shown in 4.

A Vernacular command—`Show Conjectures`—again provides the solution in the absence of a SerAPI query. This command lists the names of currently stated but unproved conjectures and by all observations is guaranteed to list the conjecture actively being proved first, though no such guarantee is stated in its documentation. We rely upon this presumed order to identify the name of the current conjecture, accumulating proof steps in stacks associated with each open conjecture.

The accuracy of this approach depends upon the assumption that no conjecture fails to enter proof mode after its initial sentence is executed. The only known exceptions to this rule comprise Programs, which do not enter proof mode until their first Obligation's proof is begun. Behavior due to violations of this rule is untested as all known plugins are compliant.

Special handling is required to associate each Obligation with the correct Program since

```
Require Coq.Program.Tactics.
Set Nested Proofs Allowed.
Program Definition foo := let x := _ : unit in _ : x = tt.
(* Start first obligation of foo *)
Next Obligation.
(* Interject with new conjecture. *)
Definition foobar : unit.
Proof.
exact tt.
(* Switch back to first obligation of foo *)
Next Obligation.
exact tt.
Qed.
(* Finish proof of foobar *)
Defined.
(* Start next obligation of foo *)
Next Obligation.
simpl; match goal with |- ?a = _ => now destruct a end.
Qed.
(* foo is defined *)
```

Figure 4 A simple example showing that proofs may be interleaved and that multiple proofs (obligations) may be associated with one term.

`Show Conjectures` reveals a unique name for each Obligation. These Obligation names are reliably constructed from the Program's name as `<program_name>_obligation_<id>`, where `id` is an integer. We can thus easily accumulate proof stacks for each Obligation and maintain their association with one another. However, the special handling means any grammar extension that defines its own Obligation or Program equivalents (i.e., multi-block proofs) cannot be serialized to the same level of accuracy. If any extension does so, then each Obligation is expected to be serialized as an unrelated theorem.

We rely upon detection of definitions to determine when and if a conjecture was proved, assuming that no conjecture emits an identifier before it is defined (i.e., before it is proved). More precisely, neither verbose user feedback nor `Print All` command should indicate the conjecture is defined before its proof(s) are complete. The only allowed exceptions to this rule comprise subproofs (generally delimited by braces and bullets) whose identifiers are similarly structured to those of Obligations and are filtered with a regular expression. Due to nesting, one cannot assume that a detected definition in the midst of a proof corresponds to the conjecture. Nor can one assume that the name of the conjecture once defined will actually match its name as returned by `Show Conjectures` during the proof since Vernacular commands such as `Save <name>` exist, which introduce the conjecture into the environment under the name given.

We ultimately detect the completion of a proof by requiring two conditions: a change in the currently detected conjecture and the detection of a new definition. This rule necessarily invokes an additional assumption: a change in the current conjecture implies that either a new proof has begun or the current proof has ended (but not both). Mutual exclusivity follows from the mutually exclusive classification of Vernacular commands into proof starters or enders within Coq's source code[6]. Since we assume that a conjecture cannot emit an identifier before it is completed, we deduce that the emission of an identifier upon the change of the current conjecture implies the completion of the prior conjecture.

Finally, if the conjecture is aborted, then it will never be detected as a definition at all even though its proof has ended. Detecting an aborted proof requires a different approach for which no alternative other than explicitly checking the type of command was forthcoming. We thus assume that no grammar extension defines its own `Abort` or `Abort All` equivalents, i.e., no other type of command concludes a proof without emitting an identifier.

Limiting Resources

Efficiently serializing the files of each project and commit in the dataset requires robust automation and multiprocessing infrastructure. TODO: Wesley: Mention project that consumed 100s of GBs of RAM and several hours: coqrel, RelDefinitions.v, 532be10aa3d14a5a775c99926b813729011f54cf, 45 GB after 5 minutes, over 200 GB some time after (crashed entire extraction process) TODO: Incorporate link to open issue about Coq memory usage: https://github.com/coq/coq/issues/12487 TODO: Speak to solution to limiting time and memory: subprocess.run with timeout arg and preexec_fn arg with resource RLIMIT_AS TODO: Speak to solution to CPU limitation: semaphore to limit number of files being serialized TODO: Speak to SwiM resource limitations as well including LRU cache; or not, seems to be covered in Section 4.2

Serialization and Version Changes

SerAPI was in theory supposed to help with some of the proof assistant versioning problems mentioned in Section 5.1. In practice, though, SerAPI itself depends on the version of Coq, and we found we had to break the SerAPI abstraction barrier often as the Coq version changed. Some useful features that exist in newer versions of SerAPI are not backported to previous versions. Furthermore, while SerAPI provides a convenient interface to expose certain Coq internals, those internals are not necessarily stable. For example, SerAPI had "can't-fix" bugs involving nested proofs because the serialization errors occur in the Coq codebase itself.[7] There seems to be an implicit assumption that one is always using the latest version of Coq, which made it challenging to collect data for older versions.

TODO: integrate any remaining parsing points:

- Different representations, sometimes you want both and need to query for more info like paths (definitely came up in Passport, too) TODO: This may be indirectly addressed in the Calculating Command Dependencies section

5.3.2 Other Proof Assistants

TODO: What other proof assistants do. Here citing PISA is a good call? maybe SerAPI is nice but library on top of it is needed? at least Coq has a parser though, Isabelle really doesn't have a canonical parser and that's a huge issue ... Parsing is *even harder* in Isabelle, though things like PISA have helped with this a lot, especially for non-experts

[6] See the `vernac_classification` type.
[7] `https://github.com/ejgallego/coq-serapi/issues/117`

5.3.3 Recommendations

TODO: What the community can do to make things easier going forward. In some sense SerAPI also kind of assumes you're a PL expert, but for ML we may want a very different interface, since folks writing ML datasets and tools are often not PL experts. Or maybe we want to make ML more accessible to PL experts, too. TODO: Exposure of Vernac classification in SerAPI (VtStartProof, VtProofStep, etc.) would help substantially. In particular, we believe that 100% accurate serialization requires access to Coq's Vernacular classification functions that determine the class (e.g., proof starter, proof ender, side-effect, etc.) of a given Vernacular command, whether the command be built-in or a grammar extension. Structure the language to be machine-readable human/compiler-out-of-the-loop or at least provide a public API for parsing it into its major components (for example, C provides braces, Python requires indentation, etc.). To this end, serapi is "still a research, experimental project, and it is expected to evolve considerably"[8], and there are already ongoing plans for further developments that would expose new features that tackle some existing challenges. For instance, there is recently a plan to rebase serapi on a different project[9], which exposes features like document overviews that appear to list all the definitions in the file and the ability to fold proofs, implying that it has the capability to list theorems and gather the associated lines– one of our current challenges.

6 Related Work

We describe related work in datasets and benchmark suites for programs and proofs, in proof repair, and in machine learning for proofs.

Datasets & Benchmark Suites

The REPLica [41] user study collected incremental edit data from eight proof engineers over the course of a month. The data collected for REPLica was more incremental than the data we collected via Git commits. On the other hand, due to the difficulty in finding volunteers for the user study, the amount of data collected was too small for many modern data-hungry machine-learning tools. In contrast, our dataset is less incremental but much larger. The REPLica data may make a useful supplement to our current dataset in the future, especially when more incremental data is useful.

There have been a number of datasets and benchmark suites released recently targeting *autoformalization*: the automatic translation of natural language specifications and proofs to formal specifications and proofs in a proof assistant. Recent autoformalization datasets consisting of aligned natural and formal language examples include ProofNet [9] for Lean, and the Isabelle Parallel Corpus [14] for Isabelle/HOL. The MiniF2F [54] benchmark suites includes a number of math Olympiad problems formalized across different proof assistants, and is used to measure success on both autoformalization and proof synthesis tasks.

There are a few datasets, benchmark suites, and proving environments specifically for proof synthesis tasks. Examples include CoqGym [51] for Coq and HOList [11] for HOL Light. The distinguishing feature of our dataset is that it includes repair examples rather than static data. We hope to expand on CoqGym in the near future using static data from the latest versions of projects in our repair dataset.

[8] https://github.com/ejgallego/coq-serapi/issues/252
[9] https://github.com/ejgallego/coq-serapi/issues/252#issuecomment-1365510329

There is a large amount of work on machine learning for code in recent years, summarized for example in a recent survey paper on neurosymbolic programming [15]. We expect there is much to be learned from the lessons, infrastructure, and metrics designed for machine learning for code, especially given the similarities between code and proofs. For example, perhaps distance metrics like CodeBLEU [37] will be reasonable proxies for proximity to correctness in machine learning tasks for formal proof.

In the field of software engineering, accessible datasets facilitate new research. For example, Defects4 [24] is a collection of bugs and patches in Java. It has been frequently used as a benchmark for automated program repair [19, 46, 28], even recently, despite nearly a decade having passed since its release. We hope that producing an accessible proof repair dataset will spur new research in proof repair. We also hope that, by focusing on good benchmarks and metrics for success early on, we can avoid some of the methodology challenges faced in program repair [35].

Proof Repair

The machine-learning task that our dataset and benchmark suite focuses on is proof repair, which is summarized in the namesake thesis [38]. There is not yet published work we are aware of for machine learning for proof repair, though we are aware of ongoing machine learning for proof repair projects by other teams in parallel in proof assistants other than Coq. We plan to train and evaluate at least two distinct proof repair models in Coq using this dataset, and we hope that this dataset makes it easy for others to do the same.

Proof repair is closely related to work in proof reuse [20, 43, 13], proof refactoring [49, 48, 44], and proof transformation [33]. These and other related topics in proof engineering have a long history, described in detail in the proof engineering survey paper QED at Large [39], as well as in the proof repair namesake thesis [38].

Proof repair can be viewed as program repair [29, 22] for proofs. There is a large amount of recent work on learning to repair programs, both symbolically (for example, by way of anti-unification in Getafix [10]) and neurally (for example, by way of unsupervised learning in Break-It-Fix-It [52]).This recent work may provide useful insights when building machine learning datasets, benchmark suites, and models for proof repair, though care must be taken to consider the differences between typical programs and formal proof developments [38].

Machine Learning for Proofs

Advances in machine learning have had a transformative effect on many fields, and theorem provers are not excluded. Examples of recent work on machine learning for synthesizing formal proofs include GPT-f [34] and HTPS [26] for Metamath and Lean; Proverbot9001 [45], ASTactic [51], Tactician [12], and DIVA [21] for Coq; and DeepHOL [11] for HOL Light. Also of note is recent work on autoformalization in Isabelle/HOL [50], Lean [9], and Coq [16]. More machine learning work for proofs can be found in QED at Large [39]. Our main goal is to expand the scope of tasks covered in machine learning for proofs, reaching important tasks not previously explored.

7 Conclusions & Future Work

TODO: Conclusion will go here.

Moving forward, our immediate plan is to build machine learning models for proof repair in Coq. We would also like to develop better metrics for measuring success at repairing definitions. Finally, we hope to work with the rest of the proof assistant community to address the many challenges we have highlighted, so that we may steer machine learning for proofs in the right direction.

References

1. Europroofnet. URL: `https://europroofnet.github.io/`.
2. Openai. URL: `https://openai.com/`.
3. Proof engineering, adaptation, repair, and learning for software (pearls). URL: `https://sam.gov/opp/da84366306554cc981f37f703a78c698/view`.
4. AI for Theorem Proving, 2016-2022. URL: `http://aitp-conference.org/`.
5. 2nd MATH-AI Workshop at NeurIPS'22, 2021-2022. URL: `https://mathai2022.github.io/`.
6. Beyond Bayes: Paths Towards Universal Reasoning Systems, 2022. URL: `https://beyond-bayes.github.io/`.
7. Arpan Agrawal, Emily First, Zhanna Kaufman, Tom Reichel, Shizhuo Zhang, Timothy Zhou, Alex Sanchez-Stern, and Talia Ringer. Proofster. URL: `https://www.alexsanchezstern.com/papers/proofster.pdf`.
8. Emilio Jesús Gallego Arias. SerAPI: Machine-Friendly, Data-Centric Serialization for COQ. Technical Report hal-01384408, HAL, 2016. URL: `http://dml.mathdoc.fr/item/hal-01384408/`.
9. Zhangir Azerbayev, Bartosz Piotrowski, and Jeremy Avigad. Proofnet: A benchmark for autoformalizing and formally proving undergraduate-level mathematics problems. In *Second MATH-AI Workshop*, 2022. URL: `https://mathai2022.github.io/`.
10. Johannes Bader, Andrew Scott, Michael Pradel, and Satish Chandra. Getafix: Learning to fix bugs automatically. *Proc. ACM Program. Lang.*, 3(OOPSLA), oct 2019. `doi:10.1145/3360585`.
11. Kshitij Bansal, Sarah M Loos, Markus N Rabe, Christian Szegedy, and Stewart Wilcox. Holist: An environment for machine learning of higher order logic theorem proving. In *ICML*, 2019.
12. Lasse Blaauwbroek, Josef Urban, and Herman Geuvers. The tactician: A seamless, interactive tactic learner and prover for coq. In *Intelligent Computer Mathematics: 13th International Conference, CICM 2020, Bertinoro, Italy, July 26–31, 2020, Proceedings*, page 271–277, Berlin, Heidelberg, 2020. Springer-Verlag. `doi:10.1007/978-3-030-53518-6_17`.
13. Olivier Boite. Proof reuse with extended inductive types. In *Theorem Proving in Higher Order Logics: 17th International Conference, TPHOLs 2004, Park City, Utah, USA, September 14-17, 2004. Proceedings*, pages 50–65, Berlin, Heidelberg, 2004. Springer. `doi:10.1007/978-3-540-30142-4_4`.
14. Anthony Bordg, Yiannos A Stathopoulos, and Lawrence C Paulson. A parallel corpus of natural language and isabelle artefacts. In *7th Conference on Artificial Intelligence and Theorem Proving (AITP)*, 2022. URL: `http://aitp-conference.org/2022/abstract/AITP_2022_paper_8.pdf`.
15. Swarat Chaudhuri, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama, Yisong Yue, et al. Neurosymbolic programming. *Foundations and Trends® in Programming Languages*, 7(3):158–243, 2021.
16. Garett Cunningham, Razvan C. Bunescu, and David Juedes. Towards autoformalization of mathematics and code correctness: Experiments with elementary proofs, 2023. URL: `https://arxiv.org/abs/2301.02195`, `doi:10.48550/ARXIV.2301.02195`.
17. Marco Cuturi. Sinkhorn Distances: Lightspeed Computation of Optimal Transport. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2292–2300. Curran Associates, Inc., 2013. URL: `http://papers.nips.cc/paper/4927-sinkhorn-distances-lightspeed-computation-of-optimal-transport.pdf`.

18 Michel Marie Deza and Elena Deza. *Encyclopedia of Distances.* Springer Berlin Heidelberg, Berlin, 3 edition, October 2014. URL: https://link.springer.com/book/10.1007/978-3-642-30958-8.

19 Thomas Durieux, Matias Martinez, Martin Monperrus, Romain Sommerard, and Jifeng Xuan. Automatic repair of real bugs: An experience report on the defects4j dataset. *CoRR*, abs/1505.07002, 2015. URL: http://arxiv.org/abs/1505.07002, arXiv:1505.07002.

20 Amy Felty and Douglas Howe. Generalization and reuse of tactic proofs. In *Logic Programming and Automated Reasoning: 5th International Conference*, LPAR '94, pages 1–15, Berlin, Heidelberg, 1994. Springer. doi:10.1007/3-540-58216-9_25.

21 Emily First and Yuriy Brun. Diversity-driven automated formal verification. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)(22–27). Pittsburgh, PA, USA. https://doi. org/10.1145/3510003.3510138*, 2022.

22 Luca Gazzola, Daniela Micucci, and Leonardo Mariani. Automatic software repair: A survey. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, page 1219, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3180155.3182526.

23 Albert Jiang, Wenda Li, Jesse Han, and Wu Yuhuai. Lisa: Language models of isabelle proofs. In *6th Conference on Artificial Intelligence and Theorem Proving (AITP)*, 2021.

24 René Just, Darioush Jalali, and Michael D. Ernst. Defects4J: A Database of existing faults to enable controlled testing studies for Java programs. In *ISSTA 2014, Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 437–440, San Jose, CA, USA, July 2014. Tool demo.

25 Daniel Kühlwein, Jasmin Christian Blanchette, Cezary Kaliszyk, and Josef Urban. Mash: Machine learning for sledgehammer. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, pages 35–50, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

26 Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. Hypertree proof search for neural theorem proving. *arXiv preprint arXiv:2205.11491*, 2022.

27 Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. Hypertree proof search for neural theorem proving, 2022. URL: https://arxiv.org/abs/2205.11491, doi:10.48550/ARXIV.2205.11491.

28 Thibaud Lutellier, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. Coconut: Combining context-aware neural translation models using ensemble for program repair. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2020, page 101–114, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3395363.3397369.

29 Martin Monperrus. Automatic software repair: A bibliography. *ACM Comput. Surv.*, 51(1), jan 2018. doi:10.1145/3105906.

30 James Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957. Publisher: Society for Industrial and Applied Mathematics. URL: https://www.jstor.org/stable/2098689.

31 Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics. doi:10.3115/1073083.1073135.

32 Ofir Pele and Michael Werman. Fast and robust earth mover's distances. In *2009 IEEE 12th International Conference on Computer Vision*, pages 460–467, 2009. doi:10.1109/ICCV.2009.5459199.

33 Frank Pfenning. *Proof Transformations in Higher-Order Logic.* PhD thesis, Carnegie Mellon University Pittsburgh, 1987.

34 Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393, 2020. URL: `https://arxiv.org/abs/2009.03393`, `arXiv:2009.03393`.

35 Zichao Qi, Fan Long, Sara Achour, and Martin Rinard. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, page 24–36, New York, NY, USA, 2015. Association for Computing Machinery. `doi:10.1145/2771783.2771791`.

36 Markus N. Rabe and Christian Szegedy. Towards the automatic mathematician. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction – CADE 28*, pages 25–37, Cham, 2021. Springer International Publishing.

37 Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *CoRR*, abs/2009.10297, 2020. URL: `https://arxiv.org/abs/2009.10297`, `arXiv:2009.10297`.

38 Talia Ringer. *Proof Repair*. PhD thesis, University of Washington, 2021.

39 Talia Ringer, Karl Palmskog, Ilya Sergey, Milos Gligoric, and Zachary Tatlock. QED at large: A survey of engineering of formally verified software. *CoRR*, abs/2003.06458, 2020. URL: `https://arxiv.org/abs/2003.06458`, `arXiv:2003.06458`.

40 Talia Ringer, RanDair Porter, Nathaniel Yazdani, John Leo, and Dan Grossman. Proof repair across type equivalences. PLDI 2021, page 112–127, New York, NY, USA, 2021. Association for Computing Machinery. `doi:10.1145/3453483.3454033`.

41 Talia Ringer, Alex Sanchez-Stern, Dan Grossman, and Sorin Lerner. REPLica: REPL instrumentation for Coq analysis. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, page 99–113, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3372885.3373823`.

42 Talia Ringer, Nathaniel Yazdani, John Leo, and Dan Grossman. Adapting proof automation to adapt proofs. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2018, page 115–129, New York, NY, USA, 2018. Association for Computing Machinery. `doi:10.1145/3167094`.

43 Talia Ringer, Nathaniel Yazdani, John Leo, and Dan Grossman. Ornaments for proof reuse in coq. In *Interactive Theorem Proving*, 2019.

44 Valentin Robert. *Front-end tooling for building and maintaining dependently-typed functional programs*. PhD thesis, UC San Diego, 2018.

45 Alex Sanchez-Stern, Yousef Alhessi, Lawrence Saul, and Sorin Lerner. Generating correctness proofs with neural networks, 2019. URL: `https://arxiv.org/abs/1907.07794`, `doi:10.48550/ARXIV.1907.07794`.

46 Ming Wen, Junjie Chen, Rongxin Wu, Dan Hao, and Shing-Chi Cheung. Context-aware patch generation for better automated program repair. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, page 1–11, New York, NY, USA, 2018. Association for Computing Machinery. `doi:10.1145/3180155.3180233`.

47 Makarius Wenzel. Isabelle/jedit — a prover IDE within the PIDE framework. *CoRR*, abs/1207.3441, 2012. URL: `http://arxiv.org/abs/1207.3441`, `arXiv:1207.3441`.

48 Iain Johnston Whiteside. *Refactoring proofs*. PhD thesis, University of Edinburgh, November 2013. URL: `http://hdl.handle.net/1842/7970`.

49 Karin Wibergh. Automatic refactoring for agda. Master's thesis, Chalmers University of Technology and University of Gothenburg, 2019.

50 Yuhuai Wu, Albert Q Jiang, Wenda Li, Markus N Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. *arXiv preprint arXiv:2205.12615*, 2022.

51 Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning (ICML)*, Long Beach, CA, USA, 2019. URL: `http://proceedings.mlr.press/v97/yang19a/yang19a.pdf`.

52 Michihiro Yasunaga and Percy Liang. Break-it-fix-it: Unsupervised learning for program repair. In *International Conference on Machine Learning*, pages 11941–11952. PMLR, 2021.
53 John R. Zech, Marcus A. Badgeley, Manway Liu, Anthony B. Costa, Joseph J. Titano, and Eric Karl Oermann. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study. *PLOS Medicine*, 15(11):e1002683, November 2018. doi:10.1371/journal.pmed.1002683.
54 Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022. URL: https://openreview.net/forum?id=9ZPegFuFTFv.

A Dataset Project Sources

TODO: will reformat table (may need help), and may not include Github links if we'll release the dataset itself, since they make things harder to format

TODO: Need to denote what is currently in the dataset and what is not yet there because of issues

TODO: ITP submission guidelines say not to include an appendix, not sure what that means, may need to move our appendix table online or something? May need to ask conference chair.

Table 1 The complete list of projects under consideration for either repair or pretraining datasets. Note that the proof and sentence counts are estimated based upon the heuristic parsing described in TODO: TODO.

Project	Proof Count	Sentence Count	Repository URL
Abel	429	4063	https://github.com/math-comp/Abel
additions	103	1425	https://github.com/coq-contribs/additions
ails	225	4605	https://github.com/coq-contribs/ails
algebra	559	6609	https://github.com/coq-contribs/algebra
AML-Formalization	1532	43978	https://github.com/harp-project/AML-Formalization
amm11262	28	1154	https://github.com/coq-contribs/amm11262
analysis	4663	39651	https://github.com/math-comp/analysis
angles	93	2013	https://github.com/coq-contribs/angles
area-method	779	10603	https://github.com/coq-contribs/area-method
argosy	304	3614	https://github.com/mit-pdos/argosy
asn1fpcoq	194	2982	https://github.com/digamma-ai/asn1fpcoq
atbr	819	11089	https://github.com/coq-community/atbr
automata	344	5833	https://github.com/coq-contribs/automata
axiomatic-abp	399	4494	https://github.com/coq-contribs/axiomatic-abp

banach_tarski	479	8608	https://github.com/roglo/banach_tarski
bbv	653	7343	https://github.com/mit-plv/bbv
bdds	232	15510	https://github.com/coq-contribs/bdds
bedrock	4423	66779	https://github.com/mit-plv/bedrock
bedrock2	1233	32668	https://github.com/mit-plv/bedrock2
bedrock-mirror-shard	1726	18771	https://github.com/gmalecha/bedrock-mirror-shard
bellantonicook	498	7515	https://github.com/davidnowak/bellantonicook
bigenough	5	43	https://github.com/math-comp/bigenough
bignums	638	9191	https://github.com/coq-community/bignums
bits	436	4895	https://github.com/coq-community/bits
buchberger	753	9705	https://github.com/coq-community/buchberger
cage	456	7425	https://github.com/gstew5/cage
Categories	519	7757	https://github.com/amintimany/Categories
category-theory	1377	15827	https://github.com/jwiegley/category-theory
cecoa	1281	33151	https://github.com/davidnowak/cecoa
celsius	211	4470	https://github.com/clementblaudeau/celsius
ceramist	427	8223	https://github.com/verse-lab/ceramist
cerise	1026	35029	https://github.com/logsem/cerise
certicoq	4350	133553	https://github.com/CertiCoq/certicoq
CertiGraph	3522	106982	https://github.com/CertiGraph/CertiGraph
cgraphs	812	16559	https://github.com/julesjacobs/cgraphs
ChargeCore	299	3750	https://github.com/jesper-bengtson/ChargeCore
checker	4	55	https://github.com/coq-contribs/checker
cheerios	83	1078	https://github.com/uwplse/cheerios
chinese	137	1769	https://github.com/coq-contribs/chinese
circuits	220	2723	https://github.com/coq-contribs/circuits
ClassicalReal	1303	48201	https://github.com/QinxiangCao/ClassicalReal
cls-coq	611	20968	https://github.com/combinators/cls-coq

color	6743	113300	https://github.com/fblanqui/color
CompCert	7835	156983	https://github.com/AbsInt/CompCert
CompCertM	1499	72820	https://github.com/snu-sf/CompCertM
CompCertR	7817	161208	https://github.com/snu-sf/CompCertR
concat	514	6417	https://github.com/coq-contribs/concat
ConCert	1581	33188	https://github.com/AU-COBRA/ConCert
constructive-geometry	116	902	https://github.com/coq-contribs/constructive-geometry
constructive-ltl	634	7621	https://github.com/jwiegley/constructive-ltl
containers	1417	12466	https://github.com/coq-contribs/containers
coq_real	843	24386	https://github.com/roglo/coq_real
coq-bitset	147	3755	https://github.com/artart78/coq-bitset
coq-ceres	122	1263	https://github.com/Lysxia/coq-ceres
Coq-Combi	3520	37470	https://github.com/math-comp/Coq-Combi
coq-compile	29	2330	https://github.com/coq-ext-lib/coq-compile
coq-cunit	0	7	https://github.com/clarus/coq-cunit
coqeal	1609	17503	https://github.com/coq-community/coqeal
coq-error-handlers	0	10	https://github.com/clarus/coq-error-handlers
coq-ext-lib	297	4842	https://github.com/coq-community/coq-ext-lib
Coq-Flow-Equivalence	376	8278	https://github.com/GaloisInc/Coq-Flow-Equivalence
coq-forcing	72	1190	https://github.com/CoqHott/coq-forcing
coq-forcing	72	1190	https://github.com/ppedrot/coq-forcing
coq-function-ninjas	0	3	https://github.com/clarus/coq-function-ninjas
coq-guarded-computational-type-theory	205	3824	https://github.com/jonsterling/coq-guarded-computational-type-theory
coq-haskell	481	5512	https://github.com/jwiegley/coq-haskell
coq-http	27	795	https://github.com/liyishuai/coq-http
coq-http2	47	943	https://github.com/liyishuai/coq-http2
coq-iterable	0	15	https://github.com/clarus/coq-iterable

coq-library-complexity	2438	43607	https://github.com/uds-psl/coq-library-complexity
coq-library-undecidability	10281	153181	https://github.com/uds-psl/coq-library-undecidability
coq-library-undecidability	10281	153181	https://github.com/uds-psl/coq-library-undecidability
coq-list-plus	0	42	https://github.com/clarus/coq-list-plus
coq-list-string	9	207	https://github.com/clarus/coq-list-string
coqoban	3	455	https://github.com/coq-community/coqoban
coq-performance-tests	170	2536	https://github.com/coq-community/coq-performance-tests
Coq-Polyhedra	1086	11773	https://github.com/Coq-Polyhedra/Coq-Polyhedra
coq-procrastination	60	506	https://github.com/Armael/coq-procrastination
coq-record-update	6	243	https://github.com/tchajed/coq-record-update
coqrel	270	1763	https://github.com/CertiKOS/coqrel
coq-robot	1505	13575	https://github.com/affeldt-aist/coq-robot
coq-simple-io	0	267	https://github.com/Lysxia/coq-simple-io
coqtail-math	2515	35411	https://github.com/coq-community/coqtail-math
coqutil	797	9516	https://github.com/mit-plv/coqutil
coq-utils	298	2820	https://github.com/arthuraa/coq-utils
Core-Erlang-Formalization	550	12170	https://github.com/harp-project/Core-Erlang-Formalization
corespec	1510	21682	https://github.com/sweirich/corespec
cours-de-coq	92	792	https://github.com/coq-contribs/cours-de-coq
cpdt-japanese	431	3152	https://github.com/cpdt-japanese/cpdt-japanese
cps	436	9211	https://github.com/takanuva/cps
crellvm	722	22052	https://github.com/snu-sf/crellvm
cryptis	448	5751	https://github.com/arthuraa/cryptis
cspec	884	11566	https://github.com/mit-pdos/cspec
ct	161	1948	https://github.com/relrod/ct

ctltctl	28	383	https://github.com/coq-contribs/ctltctl
dblib	244	1994	https://github.com/coq-community/dblib
DeepSpecDB	2719	86898	https://github.com/PrincetonUniversity/DeepSpecDB
demos	69	477	https://github.com/coq-contribs/demos
dep-map	90	1843	https://github.com/coq-contribs/dep-map
deriving	67	1567	https://github.com/arthuraa/deriving
dez	1523	20350	https://github.com/Tuplanolla/dez
dictionaries	67	845	https://github.com/coq-contribs/dictionaries
disel	844	12582	https://github.com/DistributedComponents/disel
distributed-reference-counting	954	24025	https://github.com/coq-contribs/distributed-reference-counting
domains	1738	58184	https://github.com/robdockins/domains
domain-theory	51	812	https://github.com/coq-contribs/domain-theory
dot-iris	1590	12473	https://github.com/Blaisorblade/dot-iris
engine-bench	94	1264	https://github.com/mit-plv/engine-bench
ett-to-wtt	433	13925	https://github.com/TheoWinterhalter/ett-to-wtt
euler-formula	162	6757	https://github.com/coq-contribs/euler-formula
event-struct	1039	11142	https://github.com/Event-Structures/event-struct
exceptions	3	94	https://github.com/coq-contribs/exceptions
exploring-robust-property-preservation	295	6518	https://github.com/secure-compilation/exploring-robust-property-preservation
extructures	357	2986	https://github.com/arthuraa/extructures
fcsl-pcm	2118	15806	https://github.com/imdea-software/fcsl-pcm
fermat4	130	841	https://github.com/coq-contribs/fermat4
fiat	5932	84544	https://github.com/mit-plv/fiat
finmap	870	5058	https://github.com/math-comp/finmap
finmap	870	5058	https://github.com/math-comp/finmap
float	780	10192	https://github.com/coq-contribs/float

FormalML	7026	135311	`https://github.com/IBM/FormalML`
formal-type-theory	67	10680	`https://github.com/TheoWinterhalter/formal-type-theory`
frap	1870	23058	`https://github.com/achlipala/frap`
free-groups	33	484	`https://github.com/coq-contribs/free-groups`
fssec-model	153	3255	`https://github.com/coq-contribs/fssec-model`
functional-algebra	278	1286	`https://github.com/llee454/functional-algebra`
functions-in-zfc	632	2437	`https://github.com/coq-contribs/functions-in-zfc`
fundamental-arithmetics	152	1941	`https://github.com/coq-contribs/fundamental-arithmetics`
general-type-theories	546	10172	`https://github.com/peterlefanulumsdaine/general-type-theories`
generic-environments	269	2818	`https://github.com/coq-community/generic-environments`
GeoCoq	4087	122527	`https://github.com/GeoCoq/GeoCoq`
goedel	103	8548	`https://github.com/coq-community/goedel`
GraphCoQL	220	2513	`https://github.com/imfd/GraphCoQL`
graphs	174	4460	`https://github.com/coq-contribs/graphs`
graph-theory	2272	29435	`https://github.com/coq-community/graph-theory`
groups	14	134	`https://github.com/coq-contribs/groups`
group-theory	105	1099	`https://github.com/coq-contribs/group-theory`
hahn	1458	9836	`https://github.com/vafeiadis/hahn`
hanoi	577	10514	`https://github.com/thery/hanoi`
hardware	186	1643	`https://github.com/coq-contribs/hardware`
hedges	110	2310	`https://github.com/coq-contribs/hedges`
helix	2046	56461	`https://github.com/vzaliva/helix`
higman-cf	30	279	`https://github.com/coq-contribs/higman-cf`
higman-s	49	913	`https://github.com/coq-contribs/higman-s`
hoare-tut	25	346	`https://github.com/coq-community/hoare-tut`
HoTT	4974	62461	`https://github.com/HoTT/HoTT`

HoTT-categories	380	4559	https://github.com/CategoricalData/HoTT-categories
htt	410	4143	https://github.com/imdea-software/htt
huffman	285	3910	https://github.com/coq-community/huffman
hybrid	524	7099	https://github.com/Eelis/hybrid
idxassoc	58	713	https://github.com/coq-contribs/idxassoc
ieee754	15	239	https://github.com/coq-contribs/ieee754
IFC	756	11435	https://github.com/QuickChick/IFC
imm	1444	24624	https://github.com/weakmemory/imm
infinitary-rewriting-coq	262	4207	https://github.com/martijnvermaat/infinitary-rewriting-coq
infotheo	3335	45155	https://github.com/affeldt-aist/infotheo
InfSeqExt	145	1943	https://github.com/DistributedComponents/InfSeqExt
InteractionTrees	1822	45795	https://github.com/DeepSpec/InteractionTrees
int-map	494	7816	https://github.com/coq-contribs/int-map
ipc	479	9102	https://github.com/coq-contribs/ipc
iris-coq	4299	32669	https://github.com/izgzhen/iris-coq
iron	1519	30137	https://github.com/discus-lang/iron
izf	122	827	https://github.com/coq-contribs/izf
jordan-curve-theorem	663	42492	https://github.com/coq-contribs/jordan-curve-theorem
kami	2092	41182	https://github.com/mit-plv/kami
Kami	2120	37732	https://github.com/sifive/Kami
katamaran	854	14929	https://github.com/katamaran-project/katamaran
Ktheory	40	505	https://github.com/DanGrayson/Ktheory
lambda	90	941	https://github.com/coq-contribs/lambda
largecatmodules	467	9644	https://github.com/UniMath/largecatmodules
lazy-pcf	83	1587	https://github.com/coq-contribs/lazy-pcf
lemma-overloading	840	6476	https://github.com/coq-community/lemma-overloading
lesniewski-mereology	140	1396	https://github.com/coq-contribs/lesniewski-mereology

lin-alg	543	10797	`https://github.com/coq-contribs/lin-alg`
linearscan	304	4786	`https://github.com/jwiegley/linearscan`
llvmtwin-coq	675	34538	`https://github.com/snu-sf/llvmtwin-coq`
LOGIC	1729	25450	`https://github.com/QinxiangCao/LOGIC`
loopring-protocol2-verification	155	3718	`https://github.com/sec-bit/loopring-protocol2-verification`
lvc	3782	60531	`https://github.com/sigurdschneider/lvc`
maple-mode	20	154	`https://github.com/coq-contribs/maple-mode`
MapleS	7629	153467	`https://github.com/namefanwjcom/MapleS`
markov	31	951	`https://github.com/coq-contribs/markov`
math-classes	1741	17370	`https://github.com/coq-community/math-classes`
math-comp	14263	104712	`https://github.com/math-comp/math-comp`
maths	38	386	`https://github.com/coq-contribs/maths`
memory-safe-language	72	1325	`https://github.com/arthuraa/memory-safe-language`
metacoq	8885	161078	`https://github.com/MetaCoq/metacoq`
metalib	796	8026	`https://github.com/plclub/metalib`
micro-policies-coq	569	13727	`https://github.com/micro-policies/micro-policies-coq`
mini-compiler	4	86	`https://github.com/coq-contribs/mini-compiler`
miniml	12	681	`https://github.com/coq-contribs/miniml`
mirror-core	1380	36458	`https://github.com/gmalecha/mirror-core`
mirror-shard	636	12465	`https://github.com/gmalecha/mirror-shard`
mod-red	271	6204	`https://github.com/coq-contribs/mod-red`
monae	1661	16336	`https://github.com/affeldt-aist/monae`
multinomials	1008	7573	`https://github.com/math-comp/multinomials`
oeuf	10471	186512	`https://github.com/uwplse/oeuf`
orb-stab	40	1590	`https://github.com/coq-contribs/orb-stab`
otway-rees	21	613	`https://github.com/coq-contribs/otway-rees`
paco	1519	23749	`https://github.com/snu-sf/paco`
paramcoq-iff	285	6527	`https://github.com/aa755/paramcoq-iff`
param-pi	72	3853	`https://github.com/coq-contribs/param-pi`

parsing-parses	274	3512	https://github.com/JasonGross/parsing-parses
perennial	5284	137211	https://github.com/mit-pdos/perennial
pigeons	6	34	https://github.com/llee454/pigeons
pipcore	1637	130325	https://github.com/2xs/pipcore
pnp	356	2857	https://github.com/ilyasergey/pnp
pocklington	265	6546	https://github.com/coq-community/pocklington
PolTac	309	2247	https://github.com/thery/PolTac
probchain	246	4801	https://github.com/certichain/probchain
ProcKami	54	2055	https://github.com/sifive/ProcKami
promising-coq	835	29947	https://github.com/snu-sf/promising-coq
proofs	174	2558	https://github.com/stepchowfun/proofs
propcalc	66	629	https://github.com/coq-contribs/propcalc
pts	350	4513	https://github.com/coq-contribs/pts
puiseuxth	910	20553	https://github.com/roglo/puiseuxth
qarith	60	1151	https://github.com/coq-contribs/qarith
qarith-stern-brocot	1131	16216	https://github.com/coq-community/qarith-stern-brocot
quadcopter	1061	34482	https://github.com/dricketts/quadcopter
QuantumLib	1482	20473	https://github.com/inQWIRE/QuantumLib
QuickChick	1211	13104	https://github.com/QuickChick/QuickChick
quicksort-complexity	591	8858	https://github.com/coq-contribs/quicksort-complexity
QWIRE	1167	18932	https://github.com/inQWIRE/QWIRE
railroad-crossing	100	1108	https://github.com/coq-contribs/railroad-crossing
ramsey	8	55	https://github.com/coq-contribs/ramsey
regexp	105	1556	https://github.com/coq-contribs/regexp
regex-reexamined-coq	476	7609	https://github.com/awalterschulze/regex-reexamined-coq
rem	8	113	https://github.com/coq-contribs/rem
rewriter	1436	13713	https://github.com/mit-plv/rewriter
rezk_completion	350	4261	https://github.com/benediktahrens/rezk_completion

riscv-coq	209	3367	`https://github.com/samuelgruetter/riscv-coq`
rsa	113	1194	`https://github.com/coq-contribs/rsa`
ruler-compass-geometry	332	4376	`https://github.com/coq-contribs/ruler-compass-geometry`
SCEV-coq	23	475	`https://github.com/bollu/SCEV-coq`
schroeder	14	276	`https://github.com/coq-contribs/schroeder`
search-trees	50	593	`https://github.com/coq-contribs/search-trees`
SeLoC	381	7840	`https://github.com/co-dan/SeLoC`
Set-Theory	2703	66081	`https://github.com/choukh/Set-Theory`
shuffle	40	430	`https://github.com/coq-contribs/shuffle`
silveroak	1519	26477	`https://github.com/project-oak/silveroak`
sirtt	179	4645	`https://github.com/TheoWinterhalter/sirtt`
smc	685	25108	`https://github.com/coq-contribs/smc`
SQIR	1713	37680	`https://github.com/inQWIRE/SQIR`
SquiggleEq	1302	16922	`https://github.com/aa755/SquiggleEq`
ssprove	1298	21179	`https://github.com/SSProve/ssprove`
ssrbit	344	2966	`https://github.com/ejgallego/ssrbit`
StdLibKami	72	4505	`https://github.com/sifive/StdLibKami`
StructTact	287	3563	`https://github.com/uwplse/StructTact`
subst	417	3650	`https://github.com/coq-contribs/subst`
sudoku	250	4015	`https://github.com/coq-community/sudoku`
system	1	112	`https://github.com/coq-concurrency/system`
tarjan	459	5187	`https://github.com/math-comp/tarjan`
tarski-geometry	136	2702	`https://github.com/coq-contribs/tarski-geometry`
three-gap	81	1091	`https://github.com/coq-contribs/three-gap`
tlc	2314	24176	`https://github.com/charguer/tlc`
topology	616	16461	`https://github.com/coq-community/topology`
tortoise-hare-algorithm	4	82	`https://github.com/coq-contribs/tortoise-hare-algorithm`
toychain	264	4320	`https://github.com/certichain/toychain`
transfer	201	1726	`https://github.com/Zimmi48/transfer`

traversable-fincontainer	71	871	https://github.com/coq-contribs/traversable-fincontainer
tree-automata	834	24988	https://github.com/coq-contribs/tree-automata
twoSquare	202	1769	https://github.com/thery/twoSquare
TypeTheory	1923	30575	https://github.com/UniMath/TypeTheory
UnifySL	1193	19223	https://github.com/QinxiangCao/UnifySL
UnifySL	1193	19223	https://github.com/QinxiangCao/UnifySL
UniMath	15201	256571	https://github.com/UniMath/UniMath
univalent_parametricity	399	5121	https://github.com/CoqHott/univalent_parametricity
Valuations	184	5079	https://github.com/FFaissole/Valuations
vellvm-legacy	2484	40111	https://github.com/vellvm/vellvm-legacy
velus	3338	58852	https://github.com/INRIA/velus
verdi	618	12534	https://github.com/uwplse/verdi
verdi-chord	990	16420	https://github.com/DistributedComponents/verdi-chord
verdi-raft	2272	42557	https://github.com/uwplse/verdi-raft
Verified-FEC	994	20957	https://github.com/verified-network-toolchain/Verified-FEC
verified-ifc	884	15495	https://github.com/micro-policies/verified-ifc
VeriGHC	118	1522	https://github.com/trommler/VeriGHC
VST	27777	570848	https://github.com/PrincetonUniversity/VST
WasmCert-Coq	574	13803	https://github.com/WasmCert/WasmCert-Coq
weakestmoToImm	1282	33541	https://github.com/weakmemory/weakestmoToImm
weak-up-to	150	1720	https://github.com/coq-contribs/weak-up-to
when-good-components-go-bad	1030	23295	https://github.com/secure-compilation/when-good-components-go-bad
yalla	1019	30165	https://github.com/olaure01/yalla
ynot	885	9996	https://github.com/ynot-harvard/ynot
zchinese	43	781	https://github.com/coq-contribs/zchinese
zf	213	2977	https://github.com/coq-contribs/zf
zfc	241	2450	https://github.com/coq-contribs/zfc

zorns-lemma	183	4482	`https://github.com/coq-community/zorns-lemma`
zsearch-trees	48	639	`https://github.com/coq-contribs/zsearch-trees`
Total	**344,316**	**6,199,931**	

Table 2 The list of projects accepted for inclusion into either the repair or pretraining datasets. Note that the proof and sentence counts are estimated based upon the heuristic parsing described in TODO: TODO.

Project	**Proof Count**	**Sentence Count**	**Repository URL**
Abel	429	4063	`https://github.com/math-comp/Abel`
additions	103	1425	`https://github.com/coq-contribs/additions`
ails	225	4605	`https://github.com/coq-contribs/ails`
algebra	559	6609	`https://github.com/coq-contribs/algebra`
amm11262	28	1154	`https://github.com/coq-contribs/amm11262`
analysis	4663	39651	`https://github.com/math-comp/analysis`
angles	93	2013	`https://github.com/coq-contribs/angles`
area-method	779	10603	`https://github.com/coq-contribs/area-method`
asn1fpcoq	194	2982	`https://github.com/digamma-ai/asn1fpcoq`
atbr	819	11089	`https://github.com/coq-community/atbr`
automata	344	5833	`https://github.com/coq-contribs/automata`
axiomatic-abp	399	4494	`https://github.com/coq-contribs/axiomatic-abp`
bbv	653	7343	`https://github.com/mit-plv/bbv`
bdds	232	15510	`https://github.com/coq-contribs/bdds`
bellantonicook	498	7515	`https://github.com/davidnowak/bellantonicook`
bigenough	5	43	`https://github.com/math-comp/bigenough`
bignums	638	9191	`https://github.com/coq-community/bignums`
bits	436	4895	`https://github.com/coq-community/bits`
buchberger	753	9705	`https://github.com/coq-community/buchberger`
Categories	519	7757	`https://github.com/amintimany/Categories`
category-theory	1377	15827	`https://github.com/jwiegley/category-theory`

celsius	211	4470	https://github.com/clementblaudeau/celsius
cerise	1026	35029	https://github.com/logsem/cerise
certicoq	4350	133553	https://github.com/CertiCoq/certicoq
CertiGraph	3522	106982	https://github.com/CertiGraph/CertiGraph
cgraphs	812	16559	https://github.com/julesjacobs/cgraphs
ChargeCore	299	3750	https://github.com/jesper-bengtson/ChargeCore
checker	4	55	https://github.com/coq-contribs/checker
chinese	137	1769	https://github.com/coq-contribs/chinese
circuits	220	2723	https://github.com/coq-contribs/circuits
ClassicalReal	1303	48201	https://github.com/QinxiangCao/ClassicalReal
cls-coq	611	20968	https://github.com/combinators/cls-coq
CompCert	7835	156983	https://github.com/AbsInt/CompCert
CompCertR	7817	161208	https://github.com/snu-sf/CompCertR
concat	514	6417	https://github.com/coq-contribs/concat
ConCert	1581	33188	https://github.com/AU-COBRA/ConCert
constructive-geometry	116	902	https://github.com/coq-contribs/constructive-geometry
constructive-ltl	634	7621	https://github.com/jwiegley/constructive-ltl
containers	1417	12466	https://github.com/coq-contribs/containers
coq-ceres	122	1263	https://github.com/Lysxia/coq-ceres
coq-cunit	0	7	https://github.com/clarus/coq-cunit
coqeal	1609	17503	https://github.com/coq-community/coqeal
coq-error-handlers	0	10	https://github.com/clarus/coq-error-handlers
coq-ext-lib	297	4842	https://github.com/coq-community/coq-ext-lib
Coq-Flow-Equivalence	376	8278	https://github.com/GaloisInc/Coq-Flow-Equivalence
coq-function-ninjas	0	3	https://github.com/clarus/coq-function-ninjas
coq-haskell	481	5512	https://github.com/jwiegley/coq-haskell
coq-http	27	795	https://github.com/liyishuai/coq-http

coq-iterable	0	15	`https://github.com/clarus/coq-iterable`
coq-list-plus	0	42	`https://github.com/clarus/coq-list-plus`
coq-list-string	9	207	`https://github.com/clarus/coq-list-string`
coqoban	3	455	`https://github.com/coq-community/coqoban`
coq-performance-tests	170	2536	`https://github.com/coq-community/coq-performance-tests`
coq-procrastination	60	506	`https://github.com/Armael/coq-procrastination`
coqrel	270	1763	`https://github.com/CertiKOS/coqrel`
coq-robot	1505	13575	`https://github.com/affeldt-aist/coq-robot`
coq-simple-io	0	267	`https://github.com/Lysxia/coq-simple-io`
coqtail-math	2515	35411	`https://github.com/coq-community/coqtail-math`
Core-Erlang-Formalization	550	12170	`https://github.com/harp-project/Core-Erlang-Formalization`
cours-de-coq	92	792	`https://github.com/coq-contribs/cours-de-coq`
cpdt-japanese	431	3152	`https://github.com/cpdt-japanese/cpdt-japanese`
cryptis	448	5751	`https://github.com/arthuraa/cryptis`
ctltctl	28	383	`https://github.com/coq-contribs/ctltctl`
dblib	244	1994	`https://github.com/coq-community/dblib`
demos	69	477	`https://github.com/coq-contribs/demos`
dep-map	90	1843	`https://github.com/coq-contribs/dep-map`
deriving	67	1567	`https://github.com/arthuraa/deriving`
dictionaries	67	845	`https://github.com/coq-contribs/dictionaries`
disel	844	12582	`https://github.com/DistributedComponents/disel`
distributed-reference-counting	954	24025	`https://github.com/coq-contribs/distributed-reference-counting`
domain-theory	51	812	`https://github.com/coq-contribs/domain-theory`
dot-iris	1590	12473	`https://github.com/Blaisorblade/dot-iris`
euler-formula	162	6757	`https://github.com/coq-contribs/euler-formula`

event-struct	1039	11142	`https://github.com/Event-Structures/event-struct`
exceptions	3	94	`https://github.com/coq-contribs/exceptions`
fcsl-pcm	2118	15806	`https://github.com/imdea-software/fcsl-pcm`
fermat4	130	841	`https://github.com/coq-contribs/fermat4`
finmap	870	5058	`https://github.com/math-comp/finmap`
float	780	10192	`https://github.com/coq-contribs/float`
FormalML	7026	135311	`https://github.com/IBM/FormalML`
frap	1870	23058	`https://github.com/achlipala/frap`
free-groups	33	484	`https://github.com/coq-contribs/free-groups`
fssec-model	153	3255	`https://github.com/coq-contribs/fssec-model`
functions-in-zfc	632	2437	`https://github.com/coq-contribs/functions-in-zfc`
fundamental-arithmetics	152	1941	`https://github.com/coq-contribs/fundamental-arithmetics`
generic-environments	269	2818	`https://github.com/coq-community/generic-environments`
GeoCoq	4087	122527	`https://github.com/GeoCoq/GeoCoq`
goedel	103	8548	`https://github.com/coq-community/goedel`
GraphCoQL	220	2513	`https://github.com/imfd/GraphCoQL`
graphs	174	4460	`https://github.com/coq-contribs/graphs`
graph-theory	2272	29435	`https://github.com/coq-community/graph-theory`
groups	14	134	`https://github.com/coq-contribs/groups`
group-theory	105	1099	`https://github.com/coq-contribs/group-theory`
hardware	186	1643	`https://github.com/coq-contribs/hardware`
hedges	110	2310	`https://github.com/coq-contribs/hedges`
higman-cf	30	279	`https://github.com/coq-contribs/higman-cf`
higman-s	49	913	`https://github.com/coq-contribs/higman-s`
hoare-tut	25	346	`https://github.com/coq-community/hoare-tut`
HoTT	4974	62461	`https://github.com/HoTT/HoTT`
htt	410	4143	`https://github.com/imdea-software/htt`

huffman	285	3910	https://github.com/coq-community/huffman
idxassoc	58	713	https://github.com/coq-contribs/idxassoc
ieee754	15	239	https://github.com/coq-contribs/ieee754
infotheo	3335	45155	https://github.com/affeldt-aist/infotheo
InfSeqExt	145	1943	https://github.com/DistributedComponents/InfSeqExt
InteractionTrees	1822	45795	https://github.com/DeepSpec/InteractionTrees
int-map	494	7816	https://github.com/coq-contribs/int-map
ipc	479	9102	https://github.com/coq-contribs/ipc
izf	122	827	https://github.com/coq-contribs/izf
jordan-curve-theorem	663	42492	https://github.com/coq-contribs/jordan-curve-theorem
katamaran	854	14929	https://github.com/katamaran-project/katamaran
lambda	90	941	https://github.com/coq-contribs/lambda
lazy-pcf	83	1587	https://github.com/coq-contribs/lazy-pcf
lemma-overloading	840	6476	https://github.com/coq-community/lemma-overloading
lesniewski-mereology	140	1396	https://github.com/coq-contribs/lesniewski-mereology
MapleS	7629	153467	https://github.com/namefanwjcom/MapleS
markov	31	951	https://github.com/coq-contribs/markov
math-classes	1741	17370	https://github.com/coq-community/math-classes
math-comp	14263	104712	https://github.com/math-comp/math-comp
maths	38	386	https://github.com/coq-contribs/maths
mini-compiler	4	86	https://github.com/coq-contribs/mini-compiler
miniml	12	681	https://github.com/coq-contribs/miniml
mod-red	271	6204	https://github.com/coq-contribs/mod-red
monae	1661	16336	https://github.com/affeldt-aist/monae
multinomials	1008	7573	https://github.com/math-comp/multinomials
otway-rees	21	613	https://github.com/coq-contribs/otway-rees
paco	1519	23749	https://github.com/snu-sf/paco

param-pi	72	3853	https://github.com/coq-contribs/param-pi
pigeons	6	34	https://github.com/llee454/pigeons
pipcore	1637	130325	https://github.com/2xs/pipcore
pnp	356	2857	https://github.com/ilyasergey/pnp
pocklington	265	6546	https://github.com/coq-community/pocklington
proofs	174	2558	https://github.com/stepchowfun/proofs
propcalc	66	629	https://github.com/coq-contribs/propcalc
pts	350	4513	https://github.com/coq-contribs/pts
qarith	60	1151	https://github.com/coq-contribs/qarith
qarith-stern-brocot	1131	16216	https://github.com/coq-community/qarith-stern-brocot
QuantumLib	1482	20473	https://github.com/inQWIRE/QuantumLib
QuickChick	1211	13104	https://github.com/QuickChick/QuickChick
quicksort-complexity	591	8858	https://github.com/coq-contribs/quicksort-complexity
QWIRE	1167	18932	https://github.com/inQWIRE/QWIRE
railroad-crossing	100	1108	https://github.com/coq-contribs/railroad-crossing
ramsey	8	55	https://github.com/coq-contribs/ramsey
regexp	105	1556	https://github.com/coq-contribs/regexp
regex-reexamined-coq	476	7609	https://github.com/awalterschulze/regex-reexamined-coq
rem	8	113	https://github.com/coq-contribs/rem
rsa	113	1194	https://github.com/coq-contribs/rsa
ruler-compass-geometry	332	4376	https://github.com/coq-contribs/ruler-compass-geometry
SCEV-coq	23	475	https://github.com/bollu/SCEV-coq
schroeder	14	276	https://github.com/coq-contribs/schroeder
search-trees	50	593	https://github.com/coq-contribs/search-trees
SeLoC	381	7840	https://github.com/co-dan/SeLoC
Set-Theory	2703	66081	https://github.com/choukh/Set-Theory
shuffle	40	430	https://github.com/coq-contribs/shuffle
sirtt	179	4645	https://github.com/TheoWinterhalter/sirtt

smc	685	25108	https://github.com/coq-contribs/smc
SQIR	1713	37680	https://github.com/inQWIRE/SQIR
SquiggleEq	1302	16922	https://github.com/aa755/SquiggleEq
ssprove	1298	21179	https://github.com/SSProve/ssprove
subst	417	3650	https://github.com/coq-contribs/subst
sudoku	250	4015	https://github.com/coq-community/sudoku
tarjan	459	5187	https://github.com/math-comp/tarjan
tarski-geometry	136	2702	https://github.com/coq-contribs/tarski-geometry
three-gap	81	1091	https://github.com/coq-contribs/three-gap
tlc	2314	24176	https://github.com/charguer/tlc
topology	616	16461	https://github.com/coq-community/topology
tortoise-hare-algorithm	4	82	https://github.com/coq-contribs/tortoise-hare-algorithm
toychain	264	4320	https://github.com/certichain/toychain
transfer	201	1726	https://github.com/Zimmi48/transfer
traversable-fincontainer	71	871	https://github.com/coq-contribs/traversable-fincontainer
tree-automata	834	24988	https://github.com/coq-contribs/tree-automata
UnifySL	1193	19223	https://github.com/QinxiangCao/UnifySL
UnifySL	1193	19223	https://github.com/QinxiangCao/UnifySL
UniMath	15201	256571	https://github.com/UniMath/UniMath
univalent_parametricity	399	5121	https://github.com/CoqHott/univalent_parametricity
velus	3338	58852	https://github.com/INRIA/velus
verdi-chord	990	16420	https://github.com/DistributedComponents/verdi-chord
weak-up-to	150	1720	https://github.com/coq-contribs/weak-up-to
zchinese	43	781	https://github.com/coq-contribs/zchinese
zf	213	2977	https://github.com/coq-contribs/zf
zfc	241	2450	https://github.com/coq-contribs/zfc
zorns-lemma	183	4482	https://github.com/coq-community/zorns-lemma

zsearch-trees	48	639	https://github.com/coq-contribs/zsearch-trees
Total	**344,316**	**6,199,931**	

Table 3 The list of projects considered but rejected for inclusion into either the repair or pretraining datasets. Projects were rejected if they did not build under Coq 8.10.2 using either the main branch or an obviously named tag that should support Coq 8.10.2. Note that the proof and sentence counts are estimated based upon the heuristic parsing described in TODO: TODO.

Project	Proof Count	Sentence Count	Repository URL
AML-Formalization	1532	43978	https://github.com/harp-project/AML-Formalization
bedrock	4423	66779	https://github.com/mit-plv/bedrock
bedrock2	1233	32668	https://github.com/mit-plv/bedrock2
bedrock-mirror-shard	1726	18771	https://github.com/gmalecha/bedrock-mirror-shard
cage	456	7425	https://github.com/gstew5/cage
cecoa	1281	33151	https://github.com/davidnowak/cecoa
color	6743	113300	https://github.com/fblanqui/color
CompCertM	1499	72820	https://github.com/snu-sf/CompCertM
Coq-Combi	3520	37470	https://github.com/math-comp/Coq-Combi
coq-compile	29	2330	https://github.com/coq-ext-lib/coq-compile
coq-forcing	72	1190	https://github.com/CoqHott/coq-forcing
coq-guarded-computational-type-theory	205	3824	https://github.com/jonsterling/coq-guarded-computational-type-theory
coq-http2	47	943	https://github.com/liyishuai/coq-http2
coqutil	797	9516	https://github.com/mit-plv/coqutil
crellvm	722	22052	https://github.com/snu-sf/crellvm
DeepSpecDB	2719	86898	https://github.com/PrincetonUniversity/DeepSpecDB
engine-bench	94	1264	https://github.com/mit-plv/engine-bench
fiat	5932	84544	https://github.com/mit-plv/fiat
formal-type-theory	67	10680	https://github.com/TheoWinterhalter/formal-type-theory
functional-algebra	278	1286	https://github.com/llee454/functional-algebra

general-type-theories	546	10172	`https://github.com/peterlefanulumsdaine/general-type-theories`
HoTT-categories	380	4559	`https://github.com/CategoricalData/HoTT-categories`
hybrid	524	7099	`https://github.com/Eelis/hybrid`
IFC	756	11435	`https://github.com/QuickChick/IFC`
infinitary-rewriting-coq	262	4207	`https://github.com/martijnvermaat/infinitary-rewriting-coq`
iris-coq	4299	32669	`https://github.com/izgzhen/iris-coq`
iron	1519	30137	`https://github.com/discus-lang/iron`
kami	2092	41182	`https://github.com/mit-plv/kami`
lin-alg	543	10797	`https://github.com/coq-contribs/lin-alg`
linearscan	304	4786	`https://github.com/jwiegley/linearscan`
llvmtwin-coq	675	34538	`https://github.com/snu-sf/llvmtwin-coq`
maple-mode	20	154	`https://github.com/coq-contribs/maple-mode`
metalib	796	8026	`https://github.com/plclub/metalib`
mirror-core	1380	36458	`https://github.com/gmalecha/mirror-core`
mirror-shard	636	12465	`https://github.com/gmalecha/mirror-shard`
orb-stab	40	1590	`https://github.com/coq-contribs/orb-stab`
paramcoq-iff	285	6527	`https://github.com/aa755/paramcoq-iff`
parsing-parses	274	3512	`https://github.com/JasonGross/parsing-parses`
perennial	5284	137211	`https://github.com/mit-pdos/perennial`
probchain	246	4801	`https://github.com/certichain/probchain`
promising-coq	835	29947	`https://github.com/snu-sf/promising-coq`
quadcopter	1061	34482	`https://github.com/dricketts/quadcopter`
rewriter	1436	13713	`https://github.com/mit-plv/rewriter`
rezk_completion	350	4261	`https://github.com/benediktahrens/rezk_completion`
ssrbit	344	2966	`https://github.com/ejgallego/ssrbit`
system	1	112	`https://github.com/coq-concurrency/system`
yalla	1019	30165	`https://github.com/olaure01/yalla`
Total	**344,316**	**6,199,931**	

Table 4 The list of projects considered but skipped due to time restrictions or build complications for inclusion into either the repair or pretraining datasets. Note that the proof and sentence counts are estimated based upon the heuristic parsing described in TODO: TODO.

Project	Proof Count	Sentence Count	Repository URL
argosy	304	3614	`https://github.com/mit-pdos/argosy`
banach_tarski	479	8608	`https://github.com/roglo/banach_tarski`
ceramist	427	8223	`https://github.com/verse-lab/ceramist`
cheerios	83	1078	`https://github.com/uwplse/cheerios`
coq_real	843	24386	`https://github.com/roglo/coq_real`
coq-bitset	147	3755	`https://github.com/artart78/coq-bitset`
coq-forcing	72	1190	`https://github.com/CoqHott/coq-forcing`
coq-library-complexity	2438	43607	`https://github.com/uds-psl/coq-library-complexity`
coq-library-undecidability	10281	153181	`https://github.com/uds-psl/coq-library-undecidability`
Coq-Polyhedra	1086	11773	`https://github.com/Coq-Polyhedra/Coq-Polyhedra`
coq-record-update	6	243	`https://github.com/tchajed/coq-record-update`
coq-utils	298	2820	`https://github.com/arthuraa/coq-utils`
corespec	1510	21682	`https://github.com/sweirich/corespec`
cps	436	9211	`https://github.com/takanuva/cps`
cspec	884	11566	`https://github.com/mit-pdos/cspec`
ct	161	1948	`https://github.com/relrod/ct`
dez	1523	20350	`https://github.com/Tuplanolla/dez`
domains	1738	58184	`https://github.com/robdockins/domains`
ett-to-wtt	433	13925	`https://github.com/TheoWinterhalter/ett-to-wtt`
exploring-robust-property-preservation	295	6518	`https://github.com/secure-compilation/exploring-robust-property-preservation`
extructures	357	2986	`https://github.com/arthuraa/extructures`
hahn	1458	9836	`https://github.com/vafeiadis/hahn`
hanoi	577	10514	`https://github.com/thery/hanoi`
helix	2046	56461	`https://github.com/vzaliva/helix`
imm	1444	24624	`https://github.com/weakmemory/imm`

Kami	2120	37732	`https://github.com/sifive/Kami`
Ktheory	40	505	`https://github.com/DanGrayson/Ktheory`
largecatmodules	467	9644	`https://github.com/UniMath/largecatmodules`
LOGIC	1729	25450	`https://github.com/QinxiangCao/LOGIC`
loopring-protocol2-verification	155	3718	`https://github.com/sec-bit/loopring-protocol2-verification`
lvc	3782	60531	`https://github.com/sigurdschneider/lvc`
memory-safe-language	72	1325	`https://github.com/arthuraa/memory-safe-language`
metacoq	8885	161078	`https://github.com/MetaCoq/metacoq`
micro-policies-coq	569	13727	`https://github.com/micro-policies/micro-policies-coq`
oeuf	10471	186512	`https://github.com/uwplse/oeuf`
PolTac	309	2247	`https://github.com/thery/PolTac`
ProcKami	54	2055	`https://github.com/sifive/ProcKami`
puiseuxth	910	20553	`https://github.com/roglo/puiseuxth`
riscv-coq	209	3367	`https://github.com/samuelgruetter/riscv-coq`
silveroak	1519	26477	`https://github.com/project-oak/silveroak`
StdLibKami	72	4505	`https://github.com/sifive/StdLibKami`
StructTact	287	3563	`https://github.com/uwplse/StructTact`
twoSquare	202	1769	`https://github.com/thery/twoSquare`
TypeTheory	1923	30575	`https://github.com/UniMath/TypeTheory`
Valuations	184	5079	`https://github.com/FFaissole/Valuations`
vellvm-legacy	2484	40111	`https://github.com/vellvm/vellvm-legacy`
verdi	618	12534	`https://github.com/uwplse/verdi`
verdi-raft	2272	42557	`https://github.com/uwplse/verdi-raft`
Verified-FEC	994	20957	`https://github.com/verified-network-toolchain/Verified-FEC`
verified-ifc	884	15495	`https://github.com/micro-policies/verified-ifc`
VeriGHC	118	1522	`https://github.com/trommler/VeriGHC`
VST	27777	570848	`https://github.com/PrincetonUniversity/VST`
WasmCert-Coq	574	13803	`https://github.com/WasmCert/WasmCert-Coq`

weakestmoToImm	1282	33541	https://github.com/weakmemory/weakestmoToImm
when-good-components-go-bad	1030	23295	https://github.com/secure-compilation/when-good-components-go-bad
ynot	885	9996	https://github.com/ynot-harvard/ynot
Total	**344,316**	**6,199,931**	

Table 5 The list of projects considered for inclusion into either the repair or pretraining datasets that have opam files and thus easily-inferred dependencies. Note that the proof and sentence counts are estimated based upon the heuristic parsing described in TODO: TODO.

Project	Proof Count	Sentence Count	Repository URL
Abel	429	4063	https://github.com/math-comp/Abel
analysis	4663	39651	https://github.com/math-comp/analysis
atbr	819	11089	https://github.com/coq-community/atbr
bellantonicook	498	7515	https://github.com/davidnowak/bellantonicook
bigenough	5	43	https://github.com/math-comp/bigenough
bits	436	4895	https://github.com/coq-community/bits
celsius	211	4470	https://github.com/clementblaudeau/celsius
cerise	1026	35029	https://github.com/logsem/cerise
certicoq	4350	133553	https://github.com/CertiCoq/certicoq
CertiGraph	3522	106982	https://github.com/CertiGraph/CertiGraph
cgraphs	812	16559	https://github.com/julesjacobs/cgraphs
ConCert	1581	33188	https://github.com/AU-COBRA/ConCert
coq-ceres	122	1263	https://github.com/Lysxia/coq-ceres
coqeal	1609	17503	https://github.com/coq-community/coqeal
coq-ext-lib	297	4842	https://github.com/coq-community/coq-ext-lib
coq-haskell	481	5512	https://github.com/jwiegley/coq-haskell
coq-http	27	795	https://github.com/liyishuai/coq-http
coq-list-string	9	207	https://github.com/clarus/coq-list-string
coqoban	3	455	https://github.com/coq-community/coqoban
coq-procrastination	60	506	https://github.com/Armael/coq-procrastination
coq-robot	1505	13575	https://github.com/affeldt-aist/coq-robot

coq-simple-io	0	267	https://github.com/Lysxia/coq-simple-io
coqtail-math	2515	35411	https://github.com/coq-community/coqtail-math
dblib	244	1994	https://github.com/coq-community/dblib
deriving	67	1567	https://github.com/arthuraa/deriving
dot-iris	1590	12473	https://github.com/Blaisorblade/dot-iris
event-struct	1039	11142	https://github.com/Event-Structures/event-struct
fcsl-pcm	2118	15806	https://github.com/imdea-software/fcsl-pcm
finmap	870	5058	https://github.com/math-comp/finmap
FormalML	7026	135311	https://github.com/IBM/FormalML
generic-environments	269	2818	https://github.com/coq-community/generic-environments
goedel	103	8548	https://github.com/coq-community/goedel
graph-theory	2272	29435	https://github.com/coq-community/graph-theory
hoare-tut	25	346	https://github.com/coq-community/hoare-tut
HoTT	4974	62461	https://github.com/HoTT/HoTT
htt	410	4143	https://github.com/imdea-software/htt
huffman	285	3910	https://github.com/coq-community/huffman
infotheo	3335	45155	https://github.com/affeldt-aist/infotheo
InfSeqExt	145	1943	https://github.com/DistributedComponents/InfSeqExt
InteractionTrees	1822	45795	https://github.com/DeepSpec/InteractionTrees
katamaran	854	14929	https://github.com/katamaran-project/katamaran
lemma-overloading	840	6476	https://github.com/coq-community/lemma-overloading
math-classes	1741	17370	https://github.com/coq-community/math-classes
math-comp	14263	104712	https://github.com/math-comp/math-comp
monae	1661	16336	https://github.com/affeldt-aist/monae
multinomials	1008	7573	https://github.com/math-comp/multinomials
pocklington	265	6546	https://github.com/coq-community/pocklington

qarith-stern-brocot	1131	16216	`https://github.com/coq-community/qarith-stern-brocot`
QuickChick	1211	13104	`https://github.com/QuickChick/QuickChick`
quicksort-complexity	591	8858	`https://github.com/coq-contribs/quicksort-complexity`
SeLoC	381	7840	`https://github.com/co-dan/SeLoC`
ssprove	1298	21179	`https://github.com/SSProve/ssprove`
sudoku	250	4015	`https://github.com/coq-community/sudoku`
tarjan	459	5187	`https://github.com/math-comp/tarjan`
tlc	2314	24176	`https://github.com/charguer/tlc`
topology	616	16461	`https://github.com/coq-community/topology`
toychain	264	4320	`https://github.com/certichain/toychain`
verdi-chord	990	16420	`https://github.com/DistributedComponents/verdi-chord`
Total	**344,316**	**6,199,931**	

Table 6 The list of projects considered for inclusion into either the repair or pretraining datasets that do not have opam files and thus do not have easily-inferred dependencies. Note that the proof and sentence counts are estimated based upon the heuristic parsing described in TODO: TODO.

Project	**Proof Count**	**Sentence Count**	**Repository URL**
additions	103	1425	`https://github.com/coq-contribs/additions`
ails	225	4605	`https://github.com/coq-contribs/ails`
algebra	559	6609	`https://github.com/coq-contribs/algebra`
AML-Formalization	1532	43978	`https://github.com/harp-project/AML-Formalization`
amm11262	28	1154	`https://github.com/coq-contribs/amm11262`
angles	93	2013	`https://github.com/coq-contribs/angles`
area-method	779	10603	`https://github.com/coq-contribs/area-method`
argosy	304	3614	`https://github.com/mit-pdos/argosy`
asn1fpcoq	194	2982	`https://github.com/digamma-ai/asn1fpcoq`
automata	344	5833	`https://github.com/coq-contribs/automata`
axiomatic-abp	399	4494	`https://github.com/coq-contribs/axiomatic-abp`
banach_tarski	479	8608	`https://github.com/roglo/banach_tarski`

bbv	653	7343	https://github.com/mit-plv/bbv
bdds	232	15510	https://github.com/coq-contribs/bdds
bedrock	4423	66779	https://github.com/mit-plv/bedrock
bedrock2	1233	32668	https://github.com/mit-plv/bedrock2
bedrock-mirror-shard	1726	18771	https://github.com/gmalecha/bedrock-mirror-shard
bignums	638	9191	https://github.com/coq-community/bignums
buchberger	753	9705	https://github.com/coq-community/buchberger
cage	456	7425	https://github.com/gstew5/cage
Categories	519	7757	https://github.com/amintimany/Categories
category-theory	1377	15827	https://github.com/jwiegley/category-theory
cecoa	1281	33151	https://github.com/davidnowak/cecoa
ceramist	427	8223	https://github.com/verse-lab/ceramist
ChargeCore	299	3750	https://github.com/jesper-bengtson/ChargeCore
checker	4	55	https://github.com/coq-contribs/checker
cheerios	83	1078	https://github.com/uwplse/cheerios
chinese	137	1769	https://github.com/coq-contribs/chinese
circuits	220	2723	https://github.com/coq-contribs/circuits
ClassicalReal	1303	48201	https://github.com/QinxiangCao/ClassicalReal
cls-coq	611	20968	https://github.com/combinators/cls-coq
color	6743	113300	https://github.com/fblanqui/color
CompCert	7835	156983	https://github.com/AbsInt/CompCert
CompCertM	1499	72820	https://github.com/snu-sf/CompCertM
CompCertR	7817	161208	https://github.com/snu-sf/CompCertR
concat	514	6417	https://github.com/coq-contribs/concat
constructive-geometry	116	902	https://github.com/coq-contribs/constructive-geometry
constructive-ltl	634	7621	https://github.com/jwiegley/constructive-ltl
containers	1417	12466	https://github.com/coq-contribs/containers
coq_real	843	24386	https://github.com/roglo/coq_real

coq-bitset	147	3755	https://github.com/artart78/coq-bitset
Coq-Combi	3520	37470	https://github.com/math-comp/Coq-Combi
coq-compile	29	2330	https://github.com/coq-ext-lib/coq-compile
coq-cunit	0	7	https://github.com/clarus/coq-cunit
coq-error-handlers	0	10	https://github.com/clarus/coq-error-handlers
Coq-Flow-Equivalence	376	8278	https://github.com/GaloisInc/Coq-Flow-Equivalence
coq-forcing	72	1190	https://github.com/CoqHott/coq-forcing
coq-function-ninjas	0	3	https://github.com/clarus/coq-function-ninjas
coq-guarded-computational-type-theory	205	3824	https://github.com/jonsterling/coq-guarded-computational-type-theory
coq-http2	47	943	https://github.com/liyishuai/coq-http2
coq-iterable	0	15	https://github.com/clarus/coq-iterable
coq-library-complexity	2438	43607	https://github.com/uds-psl/coq-library-complexity
coq-library-undecidability	10281	153181	https://github.com/uds-psl/coq-library-undecidability
coq-list-plus	0	42	https://github.com/clarus/coq-list-plus
coq-performance-tests	170	2536	https://github.com/coq-community/coq-performance-tests
Coq-Polyhedra	1086	11773	https://github.com/Coq-Polyhedra/Coq-Polyhedra
coq-record-update	6	243	https://github.com/tchajed/coq-record-update
coqrel	270	1763	https://github.com/CertiKOS/coqrel
coqutil	797	9516	https://github.com/mit-plv/coqutil
coq-utils	298	2820	https://github.com/arthuraa/coq-utils
Core-Erlang-Formalization	550	12170	https://github.com/harp-project/Core-Erlang-Formalization
corespec	1510	21682	https://github.com/sweirich/corespec
cours-de-coq	92	792	https://github.com/coq-contribs/cours-de-coq
cpdt-japanese	431	3152	https://github.com/cpdt-japanese/cpdt-japanese
cps	436	9211	https://github.com/takanuva/cps

crellvm	722	22052	`https://github.com/snu-sf/crellvm`
cryptis	448	5751	`https://github.com/arthuraa/cryptis`
cspec	884	11566	`https://github.com/mit-pdos/cspec`
ct	161	1948	`https://github.com/relrod/ct`
ctltctl	28	383	`https://github.com/coq-contribs/ctltctl`
DeepSpecDB	2719	86898	`https://github.com/PrincetonUniversity/DeepSpecDB`
demos	69	477	`https://github.com/coq-contribs/demos`
dep-map	90	1843	`https://github.com/coq-contribs/dep-map`
dez	1523	20350	`https://github.com/Tuplanolla/dez`
dictionaries	67	845	`https://github.com/coq-contribs/dictionaries`
disel	844	12582	`https://github.com/DistributedComponents/disel`
distributed-reference-counting	954	24025	`https://github.com/coq-contribs/distributed-reference-counting`
domains	1738	58184	`https://github.com/robdockins/domains`
domain-theory	51	812	`https://github.com/coq-contribs/domain-theory`
engine-bench	94	1264	`https://github.com/mit-plv/engine-bench`
ett-to-wtt	433	13925	`https://github.com/TheoWinterhalter/ett-to-wtt`
euler-formula	162	6757	`https://github.com/coq-contribs/euler-formula`
exceptions	3	94	`https://github.com/coq-contribs/exceptions`
exploring-robust-property-preservation	295	6518	`https://github.com/secure-compilation/exploring-robust-property-preservation`
extructures	357	2986	`https://github.com/arthuraa/extructures`
fermat4	130	841	`https://github.com/coq-contribs/fermat4`
fiat	5932	84544	`https://github.com/mit-plv/fiat`
float	780	10192	`https://github.com/coq-contribs/float`
formal-type-theory	67	10680	`https://github.com/TheoWinterhalter/formal-type-theory`
frap	1870	23058	`https://github.com/achlipala/frap`
free-groups	33	484	`https://github.com/coq-contribs/free-groups`

fssec-model	153	3255	https://github.com/coq-contribs/fssec-model
functional-algebra	278	1286	https://github.com/llee454/functional-algebra
functions-in-zfc	632	2437	https://github.com/coq-contribs/functions-in-zfc
fundamental-arithmetics	152	1941	https://github.com/coq-contribs/fundamental-arithmetics
general-type-theories	546	10172	https://github.com/peterlefanulumsdaine/general-type-theories
GeoCoq	4087	122527	https://github.com/GeoCoq/GeoCoq
GraphCoQL	220	2513	https://github.com/imfd/GraphCoQL
graphs	174	4460	https://github.com/coq-contribs/graphs
groups	14	134	https://github.com/coq-contribs/groups
group-theory	105	1099	https://github.com/coq-contribs/group-theory
hahn	1458	9836	https://github.com/vafeiadis/hahn
hanoi	577	10514	https://github.com/thery/hanoi
hardware	186	1643	https://github.com/coq-contribs/hardware
hedges	110	2310	https://github.com/coq-contribs/hedges
helix	2046	56461	https://github.com/vzaliva/helix
higman-cf	30	279	https://github.com/coq-contribs/higman-cf
higman-s	49	913	https://github.com/coq-contribs/higman-s
HoTT-categories	380	4559	https://github.com/CategoricalData/HoTT-categories
hybrid	524	7099	https://github.com/Eelis/hybrid
idxassoc	58	713	https://github.com/coq-contribs/idxassoc
ieee754	15	239	https://github.com/coq-contribs/ieee754
IFC	756	11435	https://github.com/QuickChick/IFC
imm	1444	24624	https://github.com/weakmemory/imm
infinitary-rewriting-coq	262	4207	https://github.com/martijnvermaat/infinitary-rewriting-coq
int-map	494	7816	https://github.com/coq-contribs/int-map
ipc	479	9102	https://github.com/coq-contribs/ipc
iris-coq	4299	32669	https://github.com/izgzhen/iris-coq

iron	1519	30137	https://github.com/discus-lang/iron
izf	122	827	https://github.com/coq-contribs/izf
jordan-curve-theorem	663	42492	https://github.com/coq-contribs/jordan-curve-theorem
kami	2092	41182	https://github.com/mit-plv/kami
Kami	2120	37732	https://github.com/sifive/Kami
Ktheory	40	505	https://github.com/DanGrayson/Ktheory
lambda	90	941	https://github.com/coq-contribs/lambda
largecatmodules	467	9644	https://github.com/UniMath/largecatmodules
lazy-pcf	83	1587	https://github.com/coq-contribs/lazy-pcf
lesniewski-mereology	140	1396	https://github.com/coq-contribs/lesniewski-mereology
lin-alg	543	10797	https://github.com/coq-contribs/lin-alg
linearscan	304	4786	https://github.com/jwiegley/linearscan
llvmtwin-coq	675	34538	https://github.com/snu-sf/llvmtwin-coq
LOGIC	1729	25450	https://github.com/QinxiangCao/LOGIC
loopring-protocol2-verification	155	3718	https://github.com/sec-bit/loopring-protocol2-verification
lvc	3782	60531	https://github.com/sigurdschneider/lvc
maple-mode	20	154	https://github.com/coq-contribs/maple-mode
MapleS	7629	153467	https://github.com/namefanwjcom/MapleS
markov	31	951	https://github.com/coq-contribs/markov
maths	38	386	https://github.com/coq-contribs/maths
memory-safe-language	72	1325	https://github.com/arthuraa/memory-safe-language
metacoq	8885	161078	https://github.com/MetaCoq/metacoq
metalib	796	8026	https://github.com/plclub/metalib
micro-policies-coq	569	13727	https://github.com/micro-policies/micro-policies-coq
mini-compiler	4	86	https://github.com/coq-contribs/mini-compiler
miniml	12	681	https://github.com/coq-contribs/miniml
mirror-core	1380	36458	https://github.com/gmalecha/mirror-core
mirror-shard	636	12465	https://github.com/gmalecha/mirror-shard

mod-red	271	6204	https://github.com/coq-contribs/mod-red
oeuf	10471	186512	https://github.com/uwplse/oeuf
orb-stab	40	1590	https://github.com/coq-contribs/orb-stab
otway-rees	21	613	https://github.com/coq-contribs/otway-rees
paco	1519	23749	https://github.com/snu-sf/paco
paramcoq-iff	285	6527	https://github.com/aa755/paramcoq-iff
param-pi	72	3853	https://github.com/coq-contribs/param-pi
parsing-parses	274	3512	https://github.com/JasonGross/parsing-parses
perennial	5284	137211	https://github.com/mit-pdos/perennial
pigeons	6	34	https://github.com/llee454/pigeons
pipcore	1637	130325	https://github.com/2xs/pipcore
pnp	356	2857	https://github.com/ilyasergey/pnp
PolTac	309	2247	https://github.com/thery/PolTac
probchain	246	4801	https://github.com/certichain/probchain
ProcKami	54	2055	https://github.com/sifive/ProcKami
promising-coq	835	29947	https://github.com/snu-sf/promising-coq
proofs	174	2558	https://github.com/stepchowfun/proofs
propcalc	66	629	https://github.com/coq-contribs/propcalc
pts	350	4513	https://github.com/coq-contribs/pts
puiseuxth	910	20553	https://github.com/roglo/puiseuxth
qarith	60	1151	https://github.com/coq-contribs/qarith
quadcopter	1061	34482	https://github.com/dricketts/quadcopter
QuantumLib	1482	20473	https://github.com/inQWIRE/QuantumLib
QWIRE	1167	18932	https://github.com/inQWIRE/QWIRE
railroad-crossing	100	1108	https://github.com/coq-contribs/railroad-crossing
ramsey	8	55	https://github.com/coq-contribs/ramsey
regexp	105	1556	https://github.com/coq-contribs/regexp
regex-reexamined-coq	476	7609	https://github.com/awalterschulze/regex-reexamined-coq
rem	8	113	https://github.com/coq-contribs/rem
rewriter	1436	13713	https://github.com/mit-plv/rewriter

rezk_completion	350	4261	`https://github.com/benediktahrens/rezk_completion`
riscv-coq	209	3367	`https://github.com/samuelgruetter/riscv-coq`
rsa	113	1194	`https://github.com/coq-contribs/rsa`
ruler-compass-geometry	332	4376	`https://github.com/coq-contribs/ruler-compass-geometry`
SCEV-coq	23	475	`https://github.com/bollu/SCEV-coq`
schroeder	14	276	`https://github.com/coq-contribs/schroeder`
search-trees	50	593	`https://github.com/coq-contribs/search-trees`
Set-Theory	2703	66081	`https://github.com/choukh/Set-Theory`
shuffle	40	430	`https://github.com/coq-contribs/shuffle`
silveroak	1519	26477	`https://github.com/project-oak/silveroak`
sirtt	179	4645	`https://github.com/TheoWinterhalter/sirtt`
smc	685	25108	`https://github.com/coq-contribs/smc`
SQIR	1713	37680	`https://github.com/inQWIRE/SQIR`
SquiggleEq	1302	16922	`https://github.com/aa755/SquiggleEq`
ssrbit	344	2966	`https://github.com/ejgallego/ssrbit`
StdLibKami	72	4505	`https://github.com/sifive/StdLibKami`
StructTact	287	3563	`https://github.com/uwplse/StructTact`
subst	417	3650	`https://github.com/coq-contribs/subst`
system	1	112	`https://github.com/coq-concurrency/system`
tarski-geometry	136	2702	`https://github.com/coq-contribs/tarski-geometry`
three-gap	81	1091	`https://github.com/coq-contribs/three-gap`
tortoise-hare-algorithm	4	82	`https://github.com/coq-contribs/tortoise-hare-algorithm`
transfer	201	1726	`https://github.com/Zimmi48/transfer`
traversable-fincontainer	71	871	`https://github.com/coq-contribs/traversable-fincontainer`
tree-automata	834	24988	`https://github.com/coq-contribs/tree-automata`
twoSquare	202	1769	`https://github.com/thery/twoSquare`
TypeTheory	1923	30575	`https://github.com/UniMath/TypeTheory`

UnifySL	1193	19223	https://github.com/QinxiangCao/UnifySL
UnifySL	1193	19223	https://github.com/QinxiangCao/UnifySL
UniMath	15201	256571	https://github.com/UniMath/UniMath
univalent_parametricity	399	5121	https://github.com/CoqHott/univalent_parametricity
Valuations	184	5079	https://github.com/FFaissole/Valuations
vellvm-legacy	2484	40111	https://github.com/vellvm/vellvm-legacy
velus	3338	58852	https://github.com/INRIA/velus
verdi	618	12534	https://github.com/uwplse/verdi
verdi-raft	2272	42557	https://github.com/uwplse/verdi-raft
Verified-FEC	994	20957	https://github.com/verified-network-toolchain/Verified-FEC
verified-ifc	884	15495	https://github.com/micro-policies/verified-ifc
VeriGHC	118	1522	https://github.com/trommler/VeriGHC
VST	27777	570848	https://github.com/PrincetonUniversity/VST
WasmCert-Coq	574	13803	https://github.com/WasmCert/WasmCert-Coq
weakestmoToImm	1282	33541	https://github.com/weakmemory/weakestmoToImm
weak-up-to	150	1720	https://github.com/coq-contribs/weak-up-to
when-good-components-go-bad	1030	23295	https://github.com/secure-compilation/when-good-components-go-bad
yalla	1019	30165	https://github.com/olaure01/yalla
ynot	885	9996	https://github.com/ynot-harvard/ynot
zchinese	43	781	https://github.com/coq-contribs/zchinese
zf	213	2977	https://github.com/coq-contribs/zf
zfc	241	2450	https://github.com/coq-contribs/zfc
zorns-lemma	183	4482	https://github.com/coq-community/zorns-lemma
zsearch-trees	48	639	https://github.com/coq-contribs/zsearch-trees
Total	**344,316**	**6,199,931**	