Can Large Language Models Verify System Software? A Case Study Using FSCQ as a Benchmark.

Jianxing Qin, Alexander Du, Danfeng Zhang, Matthew Lentz, and Danyang Zhuo

Motivation

Challenge: **System Software Verification**

- Critical infrastructure -> bugs are costly and dangerous
- Formal methods can verify correctness and prove the absence of bugs, but these proofs require substantial manual effort
- LLMs: promising code synthesis and reasoning skills, potential for proof automation

Research Question: **Can LLMs replace or augment the manual process of writing proofs for complex system software?**

Background Information

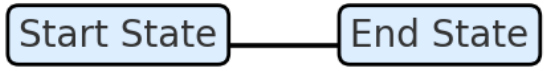
File System: A way of organizing and managing data on a disk

FSCQ (File System Certified Quick):

- A formally verified file system whose Coq-based proofs guarantee crash safety.
- Proof goals are tightly coupled with system software behavior.
  - Meets formal specifications of what we expect from it under normal execution and under any sequence of crashes, including crashes during recovery.
- Specifications are written in CHL (Crash Hoare Logic)
  - Extends Hoare-logic style (pre and postconditions) with crash conditions and recovery execution semantics
  - Embedded in Coq

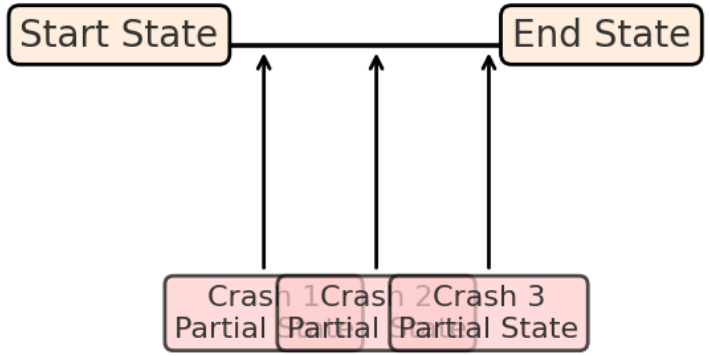
Crash-Free vs Crash Reasoning

Crash-free Reasoning



Only before & after matter

Reasoning With Crashes



Crashes expose many intermediate states
→ harder to reason about

CHL specification for FSCQ's disk_write

SPEC disk_write(a, v)

PRE **disk**: $a \mapsto \langle v_0,\ vs\rangle \star \mathit{other\_blocks}$

POST **disk**: $a \mapsto \langle v,\ [v_0] \oplus vs\rangle \star \mathit{other\_blocks}$

CRASH **disk**: $a \mapsto \langle v_0,\ vs\rangle \star \mathit{other\_blocks} \vee$
$a \mapsto \langle v,\ [v_0] \oplus vs\rangle) \star \mathit{other\_blocks}$

Related Work (Traditional Proof Automation)

Automated Theorem Provers (ATPs):

- Built on logical inference rules and heuristics.
- Automate small reasoning steps to reduce proof burden.

Interactive Theorem Provers: Coq, Isabelle, HOL Light.

  - Coq automation tactics: auto and eauto

- SMT solvers:
  - Convert proof goals into satisfiability problems
  - Examples: Dafny, Verus, and F*

Related Work (LLMs)

LLMs for Mathematical Proofs:

- Reframe theorem proving as text generation.
- Incremental production, individual tactics generated and verified
- GPT-f (2019): pioneered tactic prediction; found new shorter proofs (Metamath).
- Polu et al. (2020): improved with expert iteration and proof-size optimization, solved International Mathematical Olympiad (IMO) problems.

LLMs for Verification:

- Selene (seL4): whole-proof generation, context augmentation.
- FVEL: tactic-level, small programs.
- Rango: small finetuned LLM + linear search applied on CompCert.

Contributions

Applies off-the-shelf LLMs with best first tree search on FSCQ codebase.

Compares LLM generated proofs with original manual proofs and analyzes failing cases when LLMs cannot complete a proof.

Methodology (Best First Search for Coq)

Search Algorithm

(1) Selection

- Pick unexpanded goal with highest score.
- Score = cumulative log probability of tactics leading to it.

(2) Expansion

- Query LLM for possible next tactics.
- Each tactic:
  - Valid → completes goal or creates subgoals
  - Invalid → rejected by Coq, duplicate state, or timeout (>5s).

Search **succeeds** if all goals proven.
Search **fails** if no unexpanded goals left or query limit exceeded.

scratch FSCQ_Running_Example

```
From Coq Require Import Bool.

Lemma orb_true_r :
  forall a : bool, a || true = true.
Proof.
  intros a.
  destruct a.
  - simpl. reflexivity.
  - simpl. reflexivity.
Qed.
```

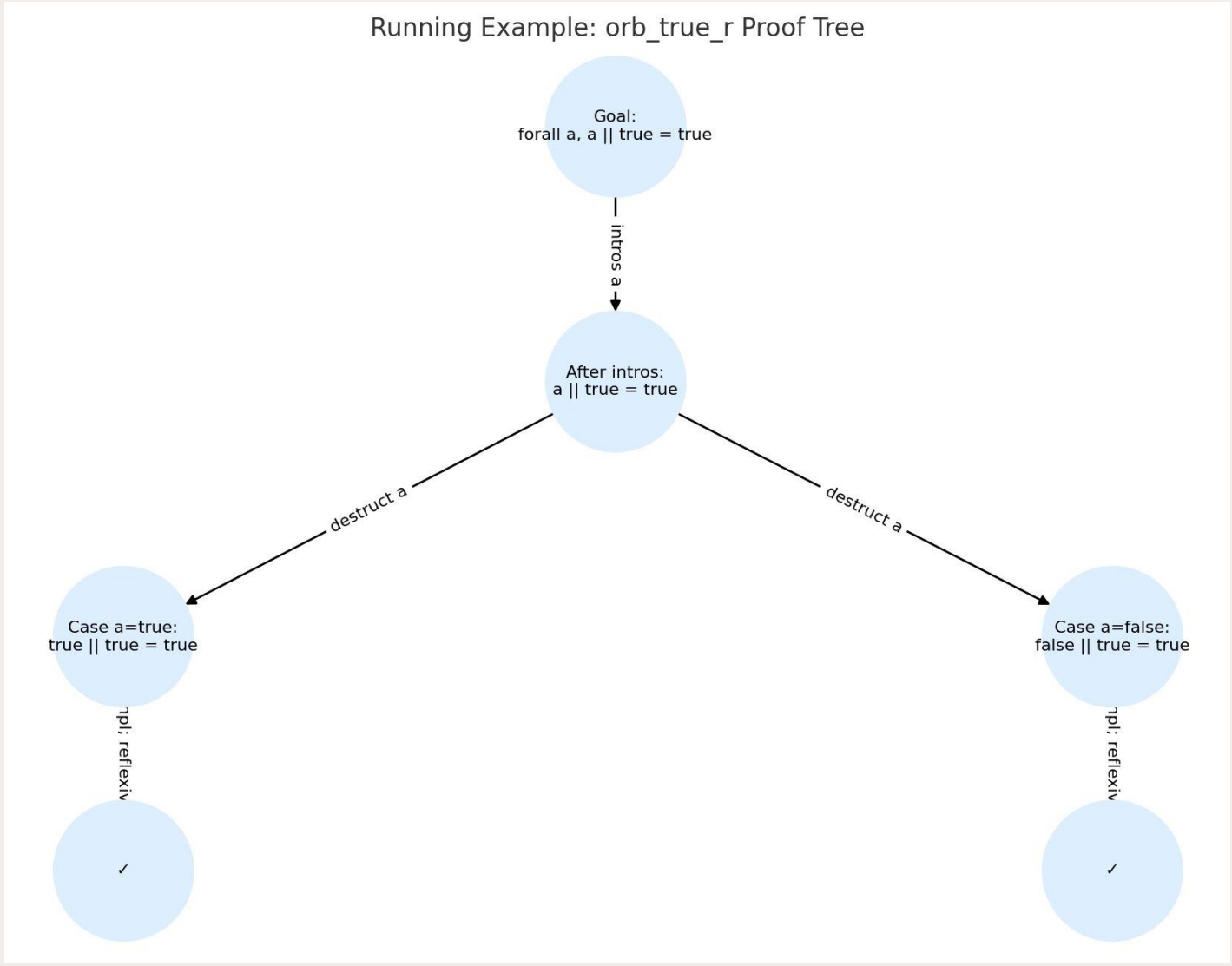
```
1 goal
_____________________________________(1/1)
forall a : bool, a || true = true
```

```
From Coq Require Import Bool.

Lemma orb_true_r :
  forall a : bool, a || true = true.
Proof.
  intros a.
  destruct a.
  - simpl. reflexivity.
  - simpl. reflexivity.
Qed.
```

```
2 goals
_____________________________________(1/2)
true || true = true
_____________________________________(2/2)
false || true = true
```

```
From Coq Require Import Bool.

Lemma orb_true_r :
  forall a : bool, a || true = true.
Proof.
  intros a.
  destruct a.
  - simpl. reflexivity.
  - simpl. reflexivity.
Qed.
```

```
All goals completed.
```


Running Example: orb_true_r Proof Tree
Goal:
forall a, a || true = true
intros a
After intros:
a || true = true
destruct a
destruct a
Case a=true:
true || true = true
Case a=false:
false || true = true
✓
✓

Methodology (Model Choices)

Evaluated four off-the-shelf LLMs:

- GPT-4o mini
- GPT-4o
- Gemini 1.5 Flash
- Gemini 1.5 Pro

Tested Gemini 1.5 Pro with two context settings:

- Full 1M-token window.
- Truncated 128k-token window.

All other models use default context limits.

- Context includes definitions, theorem statements, and proof steps in the current file and imported files up to the active proof goal
- Context too long → truncate earlier parts, keep closest tactics.

Methodology

Best-First Search Hyperparameters

- Search width: 8 (limited by Gemini's max outputs per query).
- Query limit: 128

Prompt Design

- **Vanilla setting**: proof context = only definitions + theorem statements (no proof steps)
- Hypothesis: FSCQ proofs contain repeated structural patterns → hints improve tactic prediction.
- **Hint setting**: adds human proofs for 50% of theorems (randomly chosen, fixed across runs).

Methodology

Data

- Source: theorems from FSCQ codebase.
- For smaller models (GPT-4o mini, Gemini 1.5 Flash): tested on all non-hint theorems.
- For larger models (GPT-4o, Gemini 1.5 Pro): tested on a 10% sample of non-hint theorems (~5% of all FSCQ theorems).

Evaluation Metrics

- Proof Coverage

Evaluation (Overall Proof Coverage)

Task: Measure how many FSCQ theorems LLMs can prove.

Results grouped by length of human proofs (in tokens).

Findings:

- Hinted GPT-4o solves 38% of all FSCQ theorems.
- For shorter proofs (<64 tokens) → 57% coverage (these make up ~60% of all FSCQ theorems).
- Coverage drops sharply as proofs get longer.
- No model proved theorems >512 tokens.

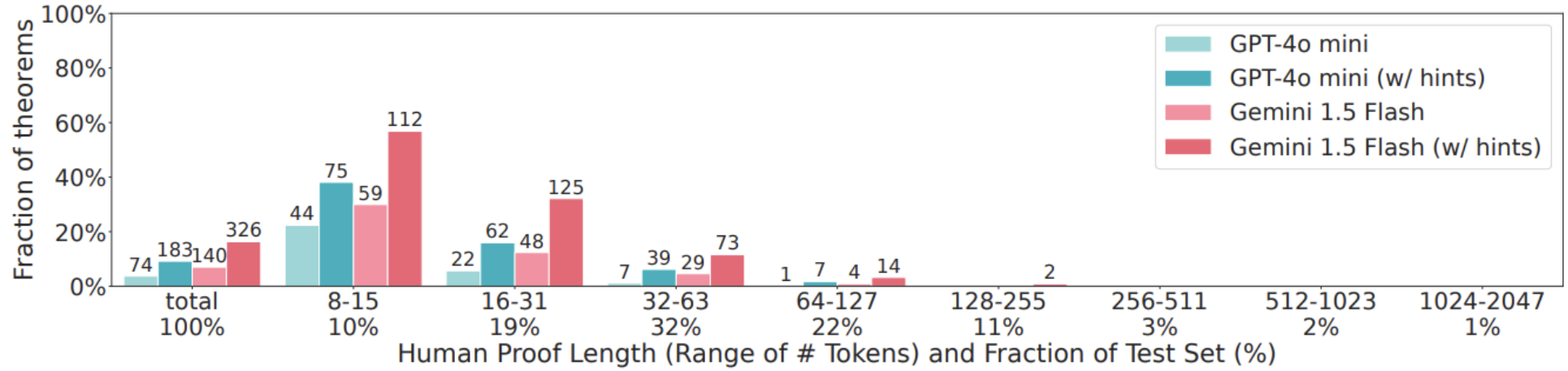
Evaluation (Hints and Context)

Hints

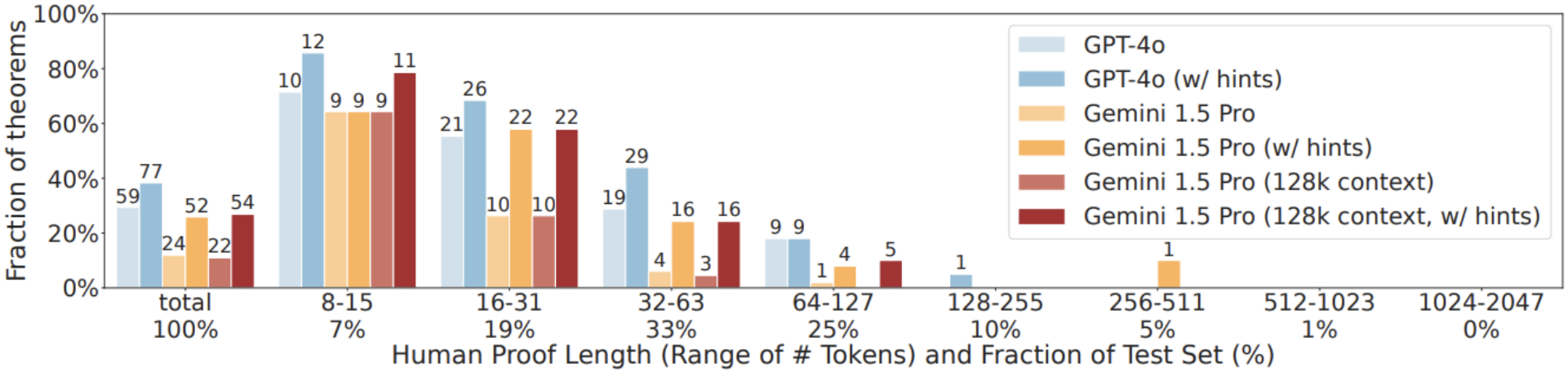
- Supplying human proofs greatly improves coverage

Context

- Gemini 1.5 Pro with 1M tokens vs 128k tokens → no improvement.
  - Suggests that more context ≠ better.
  - Implies the need for smarter context selection strategies instead of brute-force longer context.



(a) Using Small LLMs (GPT-4o mini, Gemini 1.5 Flash) on 50% of the FSCQ codebase.



(b) Using Large LLMs (GPT-4o, Gemini 1.5 Pro) on 5% of the FSCQ codebase.

Evaluation (Proof Coverage by Category)

FSCQ proofs split into 3 categories:

- Utilities – helper lemmas (general Coq libraries).
- CHL (Crash Hoare Logic) – reasoning about crash safety.
- File System – lemmas tied to real FS components.

Coverage Analysis:

- Utilities: Model performs better than expected.
- CHL: With hints, solves >50% of CHL lemmas → shows adaptation to custom proof systems.
- File System: Worse than expected, likely due to increase in dependent theorems and custom tactics

Model	Utilities	CHL	File System
GPT-4o	40.0% / 36.0%	43.3% / 32.3%	15.6% / 24.4%
GPT-4o (w/ hints)	57.8% / 46.6%	51.7% / 42.2%	20.8% / 32.0%

Reasoning vs. Memorization

Motivation: FSCQ codebase was public → risk that LLMs simply memorized proofs.
Approach: Manually compared LLM-generated proofs vs human-written proofs.
Finding: Proofs are not duplicates.

- LLMs often produce different strategies.
- Sometimes more concise than human proofs.

Similarity metric: Normalized Levenshtein distance (0 = different, 1 = identical).

- Avg similarity < 0.6, max 0.683.
- Random FSCQ proofs → similarity only 0.360.
- LLMs not memorizing FSCQ proofs.

Proof length:

- LLM proofs ≈ same token length as human proofs.
- LLMs mimic reasonable proof complexity.

[Case A] Comparison for lemma incl_tl_inv: Removing an element from the superset not in the subset preserves the subset relation.

```
Lemma incl_tl_inv : forall T l1 l2 (a : T),
incl l1 (a :: l2) -> ~ In a l1 -> incl l1 l2.
```

Original (70 Tokens)

```
induction l1; simpl; intros.
- apply incl_nil.
- intuition.
  apply incl_cons.
  + specialize (H a).
    simpl in *. intuition.
    exfalso; eauto.
  + eapply IHl1; eauto.
    eapply incl_cons_inv;
    eauto.
```

GPT-4o (67 Tokens)

```
intros T l1 l2 a H H0.
unfold incl in *.
intros x H1.
specialize (H x H1).
simpl in H.
destruct H as [H | H].
- exfalso; apply H0;
  subst; assumption.
- assumption.
```

[Case B] Comparison for lemma ndata_log_padded_log: The number of entries in a log will not change if padded with NULLs.

```
Lemma ndata_log_padded_log : forall a,
ndata_log (padded_log a) = ndata_log a.
```

Original (78 Tokens)

```
unfold ndata_log, padded_log,
setlen, roundup; intros.
rewrite firstn_oob by auto.
repeat rewrite map_app.
rewrite repeat_map; simpl.
rewrite nonzero_addrs_app.
setoid_rewrite <- app_nil_l at 3.
rewrite nonzero_addrs_app_zeros;
auto.
```

GPT-4o (29 Tokens)

```
intros a.
unfold ndata_log.
rewrite
nonzero_addrs_padded_log.
reflexivity.
```

[Case C] Comparison for tree_name_distinct_head: Uniqueness of names in a directory implies uniqueness of names in its first sub-directory.

```
Lemma tree_name_distinct_head:
forall inum name l t, tree_names_distinct
(TreeDir inum ((name, t)::l)) ->
tree_names_distinct t.
```

Original (55 Tokens)

```
intros. destruct t.
constructor. inversion H.
rewrite map_cons in H2.
apply Forall_inv in H2.
simpl in H2. inversion H2.
constructor; eauto.
```

Gemini 1.5 Pro (24 Tokens)

```
intros.
inversion H;
auto.
inversion H2;
subst; auto.
```

Model	Proved	Failed		Qualitative Metrics	
		Stuck	Fuelout	Similarity	Length
GPT-4o mini	4.2% → 9.1%	94.8% → 90.0%	1.0% → 0.9%	0.460 → 0.582	97.4% → 113.7%
GPT-4o	**29.2% → 38.1%**	65.8% → 57.9%	5.0% → 4.0%	0.546 → 0.605	101.6% → 100.7%
Gemini 1.5 Flash	7.1% → 16.3%	91.7% → 81.7%	1.2% → 2.0%	0.529 → 0.598	100.6% → 98.7%
Gemini 1.5 Pro	11.9% → 25.7%	88.1% → 73.3%	0.0% → 1.0%	0.565 → 0.660	98.7% → 92.5%
Gemini 1.5 Pro (128k context)	10.9% → 26.7%	89.1% → 72.8%	0.0% → 0.5%	0.579 → 0.683	111.2% → 109.1%

LLM Failures

Failure modes:

- Stuck = no remaining unexpanded goals (most common).
- Fuelout = query budget exhausted (rare).

Main bottleneck is reasoning ability, not query limits.

Context selection issues:

- Prompts often too long → models fail to pick relevant lemmas.
- Even simple theorems can fail.
- Manually crafted prompts (only essential definitions) let models succeed on short proofs for previously failed theorems (<16 tokens).

Reasoning models:

- Lack of interaction with the proof assistant
- High token usage and long inference times

Limitations and Future Work

- Search and Reasoning Algorithms
  - Monte Carlo Tree Search (MCTS), Chain-of-Thought prompting, Self-Reflection, and reasoning-capable models like o1.
- Off-the-Shelf vs. Fine-Tuned LLMs
  - Fine-tuning on domain-specific verification data
- LLMs Augmenting Human Proof Effort
  - Partial proof assistance from LLMs
- Improving Context Retrieval
  - More relevant and targeted context
- Constructing intermediate lemmas
- Comparing LLM efficacy across different theorem provers
- Investigate appropriate LLM evaluation methodology