Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.083290

ARTICLE

Deterministic Workflow Auditing via Dependency–State Coupled Verification and Explainable Conflict Tracing

Jixin Xu, Xingxin Li*, Senlin Zhu and Qingqing Song

Shijiazhuang Campus, Army Engineering University of PLA, Shijiazhuang, China

*Corresponding Author: Xingxin Li. Email: xingxinli_095@163.com

Received: 01 April 2026; Accepted: 05 June 2026; Published: 13 August 2026

ABSTRACT: Complex procedural workflows in maintenance and parametric design often fail due to violated long-range dependencies, unsatisfied preconditions, and inconsistent parameter bindings. In such workflows, an early structural deviation may propagate silently and eventually induce global failure. Existing workflow auditing methods, particularly those based on large language models (LLMs) or sequence matching, often rely on implicit reasoning, exhibit unstable outputs, and provide limited support for reproducible error localization. To address these limitations, we propose a verification-centric auditing framework that couples an explicit typed dependency graph with an executable state transition system. The dependency graph represents data dependencies, precondition dependencies, parameter couplings, and mutex/resource conflicts among workflow steps, while the transition system provides deterministic execution semantics for formal consistency checking. Given a structured audit instance parsed from raw or multimodal evidence, the framework verifies formal consistency through forward execution with incremental satisfiability checking over stepwise constraint stores. When an inconsistency is detected, the framework produces compact and reproducible explanations by combining provenance-guided dependency slicing, UNSAT core or near-minimal conflict-core extraction, and type-prioritized propagation-path tracing. This process identifies both conflict sources and their long-range effects. We evaluate the proposed method on two benchmarks, MyFixit-Audit (text and image, 25,000 samples) and Fusion360-Audit (text, image, and structure, 12,000 samples), covering binary consistency auditing, inconsistency typing, error localization, and structural trace recovery. Experimental results show that the proposed verifier outperforms multimodal fusion and LLM-based baselines, achieving accuracy values of 0.953 on MyFixit-Audit and 0.951 on Fusion360-Audit. In addition, it provides fully deterministic decisions, interpretable explanations, and efficient execution for reliable workflow auditing, with a median runtime of 123 ms per sample.

KEYWORDS: Workflow auditing; formal consistency checking; dependency graph; constraint solving; explainable AI; multimodal understanding

1 Introduction

Complex procedural workflows arise ubiquitously in real-world maintenance, laboratory protocols, and parametric design environments, where correctness depends on more than local step semantics. Unlike static recognition or semantic similarity matching, workflow auditing requires reasoning over structural dependencies among steps, precondition constraints, and the dynamic evolution of object states under execution semantics. This requirement is widely reflected in procedural and instructional understanding benchmarks that emphasize long-horizon state changes and compositional step structure, as well as multimodal grounding settings where text must align with visual evidence across time and steps [1–3]. In

parametric CAD, workflow steps are further coupled by explicit geometric/constraint relations and program-like design histories, making cross-step bindings and constraint satisfaction first-class signals rather than optional priors [4,5].

In practice, workflow failures are often caused not by a single step being "semantically wrong," but by implicit dependencies being violated or state constraints not being satisfied, such as missing prerequisite operations, inconsistent parameter bindings, or latent resource conflicts. These inconsistencies typically exhibit long-range error propagation: a small deviation early in the workflow may remain locally unnoticed yet amplify through subsequent state transitions and eventually lead to global failure. From an auditing perspective, practical systems further require reproducible decisions and solver-grade diagnostics. Given identical inputs, an auditor should return the same formal consistency decision, pinpoint the responsible step, element, or relation, and provide checkable evidence that supports repair.

Despite this need, most existing approaches to workflow assessment fall into two paradigms. The first relies on language-based evaluation, where a model generates or judges procedural correctness through chain-of-thought reasoning, self-consistency, or self-refinement prompts. While empirically useful, these approaches depend on implicit semantic representations and often suffer from non-determinism and unfaithful rationales, making it difficult to guarantee consistency, reproducibility, or actionable localization [6–8]. The second paradigm treats workflows as linear sequences and applies step alignment, edit distance, or sequence matching; closely related ideas also appear in process mining via log-to-model alignments and conformance metrics [9–11]. However, real workflows frequently contain commutable substructures, branching dependencies, and cross-step parameter couplings, which cannot be faithfully represented by purely sequential modeling. More broadly, both paradigms tend to conflate semantic plausibility with executability under constraints, and they lack a principled mechanism to expose how an early structural deviation propagates to a later failure, especially when constraints are induced from heterogeneous multimodal sources.

This paper revisits workflow auditing from a verification perspective and argues that formal workflow consistency is inherently a structural consistency problem under state evolution. Our design is inspired by executable verification traditions in planning and constraint solving, where formal consistency is determined by deterministic simulation combined with satisfiability checking [12,13], and by conformance checking work that emphasizes reproducible deviation diagnosis [10]. We propose a dependency–state coupled auditing framework that unifies (i) an explicit typed dependency graph capturing data, precondition, coupling, and mutex/resource relations with (ii) an executable state transition system for deterministic semantics.

We focus on deterministic auditing after workflow parsing, where the input is a structured audit instance consisting of an action sequence, parameter bindings, an initial state, and schema/template constraints. Upstream perception or parsing may be learned, but it is not the primary contribution of this work; its effect is analyzed separately in robustness experiments. Formal consistency is checked by forward execution with incremental satisfiability maintenance over step-wise constraint stores. When an inconsistency is detected, the framework produces compact and reproducible explanations via provenance-guided dependency slicing and traceability [14–17], together with principled conflict extraction based on diagnosis/UNSAT core-style minimality [18,19]. This verification-centric formulation also allows the proposed auditor to serve as a deterministic validator for learned parsers, generative workflow models, or agentic human–AI systems.

Experiments on MyFixit-Audit and Fusion360-Audit demonstrate strong gains over multimodal fusion and LLM-based baselines across binary consistency auditing, inconsistency typing, localization, and structural trace recovery, while ensuring fully deterministic decisions and efficient runtime.

2 Related Work

2.1 Workflow Auditing via Conformance Checking and Executable Verification

Workflow auditing is related to *conformance checking* in process mining, where observed traces are compared against an executable process model (often Petri nets) and deviations are diagnosed via replay or alignment [9–11,20]. Planning and executable verification provide a complementary view: a workflow is formally consistent iff it can be executed from an initial state under formal action semantics while satisfying all step-wise and cross-step constraints; plan validators such as VAL operationalize this idea for PDDL-style models [12]. SAT/SMT solvers enable deterministic decisions about formal consistency and naturally support incremental checking as constraints accumulate across steps [13,21]. Our work aligns with this verification tradition but targets workflows extracted from heterogeneous multimodal sources, requiring an explicit coupling between structural dependencies (graph) and executable state evolution (transition system).

2.2 Explainable Failure Localization: Slicing, Provenance, and UNSAT Cores

Actionable debugging has long relied on dependency analysis: program slicing isolates a compact subset relevant to a failure criterion, with dynamic variants further conditioning on a concrete execution [15]. In constraint-based verification, UNSAT cores provide a principled way to extract small conflicting constraint subsets, closely connected to diagnosis via conflicts and minimal explanations [22]. Provenance offers a formal mechanism to trace how outputs depend on inputs, originally studied in databases and later adopted broadly for traceability [16,17]. Building on these ideas, we maintain step-level provenance for state variables, slice the dependency graph at failure time, extract a (near-)minimal conflict core within the slice, and return graph-grounded propagation traces to make long-range error propagation explicit and reproducible.

2.3 Procedural Understanding, Multimodal Grounding, and LLM Graders

Procedural understanding benchmarks emphasize step-wise state changes and long-horizon dependencies [1]. Multimodal instruction grounding extends this to text–image/video alignment and step reasoning [2,3,23], while domain datasets such as repair manuals and parametric CAD sequences expose practical noise, implicit couplings, and explicit structure [4,5,24–26]. In practice, LLM-based graders using chain-of-thought, voting, and self-refinement are also used for procedural assessment, but they can be non-deterministic and their explanations may be hard to reproduce [6–8]. Even with schema-constrained outputs, format compliance does not guarantee faithful reasoning, as highlighted by structured-output benchmarks [27]. Our verifier is complementary: it allows upstream learned components (e.g., encoders or matching models) [28–31], while producing deterministic formal-consistency decisions and solver-grounded explanations (core + trace).

3 Method

3.1 Auditing Boundary and Structured Input

We separate workflow parsing from workflow verification. Let ρ denote raw evidence, such as text, images, CAD structures, or their combination. An upstream parser maps ρ to a structured audit instance:

$$g : \rho \mapsto \omega, \qquad \omega = \langle \pi, s_0, \mathcal{L}, \Gamma \rangle. \tag{1}$$

Here π is the action sequence, s_0 is the initial state, $\mathcal{L}$ is the schema and constraint-template library, and Γ stores grounding metadata such as entity identifiers, text–image links, CAD relations, or parser confidence scores. The verifier operates on ω, not directly on raw evidence. Thus, the formal guarantees below are conditional on the structured instance and library; empirical errors may additionally reflect upstream

parsing or grounding noise. Perturbations such as image swaps may occur before parsing, whereas entity, count/specification, or relation swaps can also be injected directly into ω.

Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be the state variables, and let $s \in \mathcal{S}$ be an assignment to them. Each step is a parameterized action $a_t(\theta_t)$, where $a_t \in \mathcal{A}$ and $\theta_t \in \Theta$. The workflow is

$$\pi = \langle a_1(\theta_1), \ldots, a_T(\theta_T) \rangle. \tag{2}$$

Fig. 1 illustrates the overall architecture of our verifier-in-the-loop auditing pipeline. All core functional components detailed in the following subsections are encapsulated within this unified incremental verification framework.

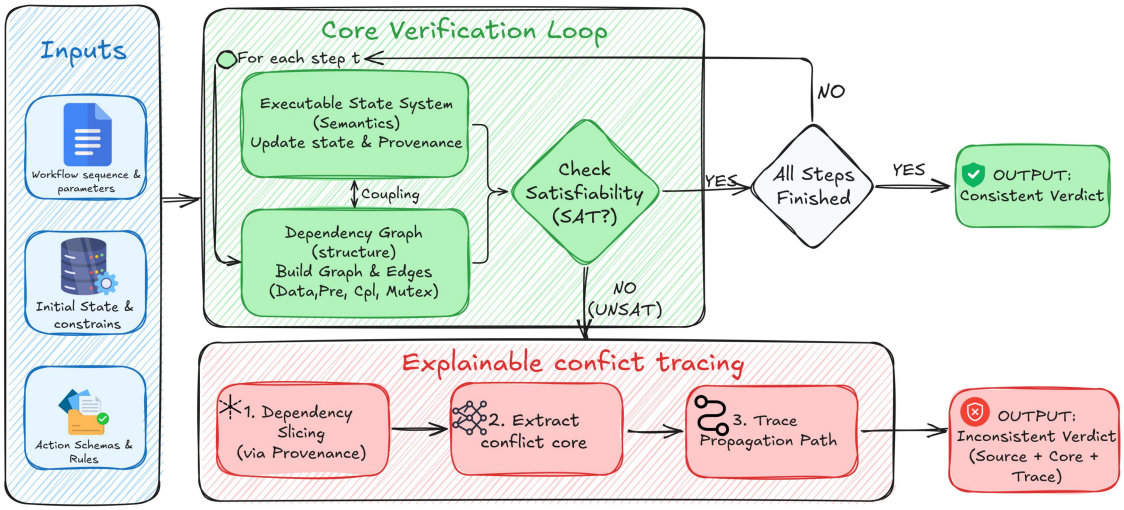


Figure 1: Verifier-in-the-loop workflow auditing pipeline. Raw or multimodal evidence is first converted by an upstream parser into a structured audit instance. The proposed verifier then instantiates action schemas and constraint templates, maintains an executable state and a typed dependency graph, and incrementally checks preconditions, invariants, parameter couplings, and mutex/resource constraints. When a violation is detected, provenance-guided slicing, conflict-core extraction, and type-prioritized path tracing produce a deterministic inconsistent verdict with reproducible explanations.

3.2 Workflow Semantics and Constraint Templates

Action schema.

Each action type is specified by

$$\mathsf{Schema}(a) = \langle R(a), W(a), \mathsf{Pre}(a), \mathsf{Eff}(a), \mathsf{Inv}(a) \rangle, \tag{3}$$

where $R(a)$ and $W(a)$ are read/write sets, $\mathsf{Pre}(a)$ gives hard preconditions, $\mathsf{Eff}(a)$ defines the state update, and $\mathsf{Inv}(a)$ gives local type, range, unit, or object-state invariants. Execution is deterministic:

$$s_t = f_{a_t}(s_{t-1}, \theta_t) \triangleq \mathsf{Apply}(\mathsf{Eff}(a_t), s_{t-1}, \theta_t). \tag{4}$$

Constraint-template library.

To clarify the source of post-conditions, we define

$$\mathcal{L} = \langle \{\mathsf{Schema}(a)\}_{a \in \mathcal{A}}, \mathcal{T}^{\mathsf{cpl}}, \mathcal{T}^{\mathsf{mutex}}, \mathcal{T}^{\mathsf{dom}} \rangle. \tag{5}$$

Schema terms provide preconditions and invariants. The template sets provide cross-step constraints: $\mathcal{T}^{\mathsf{cpl}}$ for parameter/entity bindings, $\mathcal{T}^{\mathsf{mutex}}$ for resource exclusivity or non-commutativity, and $\mathcal{T}^{\mathsf{dom}}$ for domain constraints such as repair attachment relations or CAD topology. Hence post-conditions are not an additional unexplained schema field; they are instantiated from invariants and domain templates.

For each step, constraints are instantiated as

$$\mathcal{C}_t = \mathcal{C}_t^{\mathsf{pre}} \cup \mathcal{C}_t^{\mathsf{post}}, \qquad \mathcal{C}_t^{\mathsf{post}} = \mathcal{C}_t^{\mathsf{inv}} \cup \mathcal{C}_t^{\mathsf{cpl}} \cup \mathcal{C}_t^{\mathsf{mutex}} \cup \mathcal{C}_t^{\mathsf{dom}}. \tag{6}$$

Here $\mathcal{C}_t^{\mathsf{pre}}$ is evaluated on (s_{t-1}, θ_t) before execution, while $\mathcal{C}_t^{\mathsf{post}}$ is checked on $(s_t, \theta_{\le t})$ after execution. Each grounded constraint is a decidable predicate

$$c(\cdot) \in \{\mathsf{true}, \mathsf{false}\}. \tag{7}$$

Definition 1 (Formal Workflow Consistency): *Given* $\omega = \langle \pi, s_0, \mathcal{L}, \Gamma \rangle$, *workflow* π *is* formally consistent *if there exists a trajectory* $\{s_t\}_{t=0}^{T}$ *such that, for every* t,

$$\left(\bigwedge_{c \in \mathcal{C}_t^{\mathsf{pre}}} c(s_{t-1}, \theta_t) = \mathsf{true} \right) \wedge s_t = f_{a_t}(s_{t-1}, \theta_t) \wedge \left(\bigwedge_{c \in \mathcal{C}_t^{\mathsf{post}}} c(s_t, \theta_{\le t}) = \mathsf{true} \right). \tag{8}$$

Otherwise, π is formally inconsistent. *We use consistent/inconsistent throughout this paper in this formal sense.*

3.3 Running Example

Consider a repair workflow

$$\pi_{\mathsf{ex}} = \langle \texttt{OpenCover}(D), \texttt{DisconnectBattery}(B), \texttt{RemoveBracket}(B), \texttt{LiftBattery}(B) \rangle.$$

The action $\texttt{LiftBattery}(B)$ reads $\texttt{battery\_connected}(B)$ and $\texttt{bracket\_removed}(B)$, writes $\texttt{battery\_lifted}(B)$, and requires

$$\neg \texttt{battery\_connected}(B) \wedge \texttt{bracket\_removed}(B).$$

If $\texttt{DisconnectBattery}(B)$ is omitted or moved after $\texttt{LiftBattery}(B)$, the verifier detects a precondition violation at the lift step. Provenance identifies the steps that produced the current battery state, slicing keeps only the relevant dependency subgraph, and the returned path explains how the missing or delayed disconnection causes the failure. If a later step refers to B' instead of B, a coupling template instantiates $B = B'$, and the conflict core exposes the inconsistent binding.

3.4 Explainable State and Dependency Graph

Explainable state.

We define provenance before using it in graph construction. At each step, the verifier maintains

$$\tilde{s}_t = \langle s_t, \mathsf{prov}_t \rangle, \tag{9}$$

where $\mathsf{prov}_t(x) \subseteq \{1, \ldots, t\}$ records the steps contributing to the current value of x. Initialize $\mathsf{prov}_0(x) = \varnothing$ and update

$$\mathsf{prov}_t(x) = \begin{cases} \{t\} \cup \bigcup_{y \in R(a_t)} \mathsf{prov}_{t-1}(y), & x \in W(a_t), \\ \mathsf{prov}_{t-1}(x), & \text{otherwise.} \end{cases} \tag{10}$$

Each instantiated constraint c also has a step attribution $\mathsf{attr}(c)$, namely the step or steps that introduce the constraint. We denote the state variables and parameter symbols in c by $\mathsf{vars}_X(c)$ and $\mathsf{vars}_\Theta(c)$.

Typed dependency graph.

We construct $G = (V, E, \lambda)$ with

$$V = V_A \cup V_X \cup V_\Theta, \tag{11}$$

where V_A are action nodes, V_X are state-variable nodes, V_Θ are parameter/binding nodes, and λ assigns edge types. The graph uses four edge families.

Data dependency. If a_i is the most recent writer of x before a_j reads x, add

$$(v_{a_i}, v_x), (v_x, v_{a_j}) \in E, \qquad \lambda = \mathsf{data}. \tag{12}$$

Precondition dependency. If a precondition of a_j references x, add (v_x, v_{a_j}) with $\lambda = \mathsf{pre}$. Its upstream causes are obtained from $\mathsf{prov}_{j-1}(x)$.

Coupling. For a binding between steps i and j, introduce $v_\vartheta \in V_\Theta$ and add two-way arcs between v_ϑ and the participating action nodes. The bidirectionality is for explanation traversal of symmetric equality/binding constraints; it does not imply reverse temporal causality.

Mutex/resource conflict. For a mutex or exclusive-resource template involving a_i and a_j, add an undirected conflict edge, stored as two directed arcs in implementation. If a domain rule is directional, such as a strict precedence relation, it is encoded as a directed dependency rather than as a mutex edge.

3.5 Verification and Conflict Tracing

Incremental verification.

Let Φ_t be the accumulated post-constraint store:

$$\Phi_t = \Phi_{t-1} \wedge \bigwedge_{c \in \mathcal{C}_t^{\mathsf{post}}} c(s_t, \theta_{\le t}), \qquad \Phi_0 = \top. \tag{13}$$

At step t, the verifier checks $\mathcal{C}_t^{\mathsf{pre}}$, executes $s_t = f_{a_t}(s_{t-1}, \theta_t)$ if all preconditions hold, updates provenance, adds post-constraints to Φ_t, and queries $\mathsf{SAT}(\Phi_t)$. Grounded predicates are evaluated directly; symbolic formulas are checked by a decidable solver.

Slicing and core extraction.

When a failure occurs at t^*, let $\mathcal{F}_{t^*}$ be the failed preconditions or solver-reported failing constraints. The relevant ancestor set is

$$\mathcal{U} = \bigcup_{c \in \mathcal{F}_{t^*}} \left(\mathsf{attr}(c) \cup \bigcup_{x \in \mathsf{vars}_X(c)} \mathsf{prov}_\tau(x) \right), \tag{14}$$

where $\tau = t^* - 1$ for precondition failures and $\tau = t^*$ for post-constraint failures. The sliced graph is

$$G_{t^*} = G\left[V_{t^*}^{A} \cup V_{t^*}^{X\Theta}\right], \tag{15}$$

where $V_{t^*}^{A} = \{v_{a_i} \mid i \in \mathcal{U}\} \cup \{v_{a_{t^*}}\}$ and $V_{t^*}^{X\Theta}$ contains adjacent variable/binding nodes appearing in the failure or core.

Within G_{t^*}, the verifier extracts a conflict core $\mathcal{K}$. For precondition failures, $\mathcal{K} = \mathcal{F}_{t^*}$. For post-constraint failures, $\mathcal{K} \subseteq \Phi_{t^*}$ is an UNSAT core when supported by the solver, or a deterministic near-minimal core obtained by deletion-based shrinking. The conflict sources are

$$\mathsf{Src} = \bigcup_{c \in \mathcal{K}} \left(\mathsf{attr}(c) \cup \bigcup_{x \in \mathsf{vars}_X(c)} \mathsf{prov}_{\tau}(x) \right). \tag{16}$$

Propagation paths.

For each $i \in \mathsf{Src}$, the verifier returns a path from v_{a_i} to $v_{a_{t^*}}$ in G_{t^*}. Edge costs are fixed as

$$\mathsf{cost}(e) = \begin{cases} 0, & \lambda(e) = \mathsf{pre}, \\ 1, & \lambda(e) = \mathsf{data}, \\ 2, & \lambda(e) = \mathsf{cpl}, \\ 3, & \lambda(e) = \mathsf{mutex}. \end{cases} \tag{17}$$

The selected path is

$$P_{i \to t^*} = \arg \min_{P: v_{a_i} \rightsquigarrow v_{a_{t^*}}} \left(\sum_{e \in P} \mathsf{cost}(e), \mathsf{len}(P) \right), \tag{18}$$

with lexicographic minimization and deterministic tie-breaking.

3.6 Full Procedure

Algorithm 1 formalizes the complete dependency-state coupled verification workflow. It takes a structured audit instance as input and returns a consistency verdict together with reproducible conflict explanations. The algorithm performs forward incremental verification: it checks preconditions before executing each step, maintains an accumulated constraint store, and terminates immediately upon detecting the first inconsistency. When a violation is found, it generates a compact explanation via dependency slicing, conflict core extraction, and propagation path tracing.

Algorithm 1: Dependency–state coupled workflow verification

Require: Structured instance $\omega = \langle \pi, s_0, \mathcal{L}, \Gamma \rangle$.
Ensure: Label $y \in \{\texttt{consistent}, \texttt{inconsistent}\}$; sources Src; paths $\{P\}$.
1: Build G from schemas, constraint templates, and grounding metadata.
2: Initialize $\tilde{s}_0 = \langle s_0, \mathsf{prov}_0 \rangle$ and $\Phi_0 = \top$.
3: **for** $t = 1$ to T **do**
4: Instantiate and check $\mathcal{C}_t^{\mathsf{pre}}$.
5: **if** some precondition is false **then**
6: Slice G_t, set $\mathcal{K}$ to the failed preconditions, compute Src and $\{P\}$.

(Continued)

Algorithm 1 (continued)

return `inconsistent`, Src, $\{P\}$.
end if
Execute $s_t = f_{a_t}(s_{t-1}, \theta_t)$ and update prov_t.
Instantiate $\mathcal{C}_t^{\mathsf{post}}$ and update Φ_t.
if $\mathsf{SAT}(\Phi_t) = \texttt{UNSAT}$ **then**
Slice G_t, extract $\mathcal{K}$, compute Src and $\{P\}$.
return `inconsistent`, Src, $\{P\}$.
end if
end for
return `consistent`, $\varnothing, \varnothing$.

3.7 Theoretical Properties and Complexity

We state guarantees over the structured instance ω. They do not assert perfect raw parsing; they assert deterministic verification once ω is given.

Assumption 1 *(Deterministic transitions): Each f_a is deterministic and computable.*

Assumption 2 *(Decidable constraints): Each Φ_t belongs to a decidable constraint fragment, and the solver returns correct satisfiability results.*

Assumption 3 *(Deterministic instantiation): Given ω, template instantiation returns a finite deterministic set of constraints and attributions.*

Theorem 1 *(Soundness): If Algorithm 1 outputs* `consistent`*, then π is formally consistent by Definition 1.*

Theorem 2 *(Detection completeness): If π is formally inconsistent, Algorithm 1 outputs* `inconsistent` *at the first step where a precondition is violated or Φ_t becomes unsatisfiable.*

Proposition 1 *(Traceability): For any failed constraint, (10) and $\mathsf{attr}(\cdot)$ identify a finite ancestor set; therefore the failure can be represented on G_{t^*}.*

Let T be the number of steps, $\bar{k}$ the average number of read/write variables per step, $\bar{q}$ the average number of instantiated constraints per step, and $|E|$ the number of dependency edges. Graph construction and updates cost

$$O\big(T(\bar{k} + \bar{q}) + |E|\big). \tag{19}$$

Verification is dominated by incremental solving, $\sum_{t=1}^{T} \mathsf{Solve}(\Phi_t)$. Slicing and path search are linear in the sliced graph size, $O(|E_{t^*}|)$. Core extraction requires one additional solver query when UNSAT cores are available; otherwise deterministic shrinking costs $O(m \cdot \mathsf{Solve})$ for m candidate constraints.

3.8 Domain Instantiation and Deployment Effort

The framework is domain-agnostic in form but schema-dependent in deployment. Applying it to a new dataset requires defining action types, state variables, read/write sets, preconditions, effects, invariants, coupling templates, mutex/resource templates, and the parser-to-schema mapping. This is a library-level effort; once $\mathcal{L}$ is built, new instances are verified without hand-crafted explanations.

For physical workflows, states may encode object existence, attachments, tool status, and safety/resource conditions. For CAD workflows, states may encode parameters, units, geometric relations, topological entities, and feature dependencies. The same interface can also serve as a validator for generative or agentic systems: a model or human–AI agent proposes a workflow, the verifier checks it, and the returned core and trace provide deterministic feedback for refinement. Because schemas, templates, and dependency graphs are explicit before inference, the framework also provides an ante-hoc interpretable backbone for learned components.

4 Experiments

We evaluate the proposed dependency–state coupled auditing framework on two benchmarks, **MyFixit-Audit** and **Fusion360-Audit**. The experiments are organized to validate five key capabilities: (i) binary consistency auditing, (ii) inconsistency type recognition, (iii) error localization (step/element/relation), (iv) structural propagation tracing (Fusion360-Audit), and (v) efficiency and robustness.

4.1 Datasets

Table 1 summarizes the two datasets. MyFixit-Audit is bi-modal with text and images collected from iFixit repair manuals, while Fusion360-Audit is tri-modal with text, images, and explicit assembly structure derived from Fusion 360 CAD assemblies. Both datasets follow a 70/15/15 split and are balanced between consistent and inconsistent samples. Inconsistent samples are constructed using four perturbation families, which also define the auditing labels for inconsistency type classification.

Table 1: Statistics of the auditing datasets. MyFixit-Audit is a bi-modal dataset derived from real repair manuals, while Fusion360-Audit is a tri-modal dataset based on CAD assembly data.

Statistic	MyFixit-Audit	Fusion360-Audit
Modalities	Text, Image	Text, Image, Structure
Source	iFixit Repair Manuals	Fusion 360 CAD Assemblies
Total Samples	25,000	12,000
Train	17,500 (70%)	8400 (70%)
Validation	3750 (15%)	1800 (15%)
Test	3750 (15%)	1800 (15%)
Consistency Split		
Consistent	12,500 (50%)	6000 (50%)
Inconsistent	12,500 (50%)	6000 (50%)
Inconsistency Types		
Image Swap	3750 (15%)	1200 (10%)
Entity/Text Swap	3750 (15%)	2400 (20%)
Count/Spec Perturbation	2500 (10%)	1200 (10%)
Relation/Structure Swap	2500 (10%)	1200 (10%)

Audit-instance construction and perturbation layer.

We distinguish raw-evidence perturbations from structured-instance perturbations. Each sample is first represented as raw or multimodal evidence, including text, images, and, for Fusion360-Audit, structural

CAD information. This evidence is then mapped by the same parser/schema-mapping pipeline into a structured audit instance $\omega = \langle \pi, s_0, \mathcal{L}, \Gamma \rangle$, on which the proposed verifier operates. `ImageSwap` is introduced at the raw-evidence level by replacing the visual evidence associated with a workflow step while keeping the corresponding procedural context fixed. `EntityTextSwap` and `CountSpecPerturb` modify textual entity mentions or numeric/specification fields and are then reflected in the parsed action parameters, entity bindings, or count/specification constraints. `RelationStructSwap` is applied at the structured-representation level by perturbing parsed operation relations, assembly edges, or CAD relation identifiers used by the audit instance. Therefore, the reported results evaluate a verifier operating on parsed workflows under controlled grounding and representation noise, rather than an oracle verifier with perfect symbolic inputs. This protocol separates errors caused by upstream data-to-symbol mapping from the deterministic reasoning performed by the verifier.

We report additional structural statistics used in our evaluation protocols in Table 2. Each sample is a procedural sequence of steps; Fusion360-Audit additionally contains an assembly graph.

Table 2: Additional structural statistics (procedure length and assembly graph size).

Dataset	Avg #Steps ($\bar{T}$)	Max #Steps	Avg #Nodes	Avg #Edges
MyFixit-Audit	12.4	36	–	–
Fusion360-Audit	18.7	52	34.6	58.2

4.2 Tasks and Metrics

We evaluate four tasks.

T1: Consistency Auditing (Binary). Given a sample with modalities **I**, predict $y \in \{\texttt{Consistent}, \texttt{Inconsistent}\}$. We report Accuracy and macro-F1.

T2: Inconsistency Type Classification. We perform multi-class classification over five labels: `Consistent`, `ImageSwap`, `EntityTextSwap`, `CountSpecPerturb`, and `RelationStructSwap`. We report macro-F1 and per-class F1.

T3: Error Localization. For inconsistent samples, we localize the corrupted source: (i) **step localization**, where the model predicts the corrupted step index in ranked form; we report Top-1, Top-3, Top-5, and MRR; and (ii) **element localization**, where the model predicts the corrupted element (swapped image id, swapped entity mention, or perturbed count-spec field); we report exact-match accuracy (ElemAcc). For **Fusion360-Audit** `RelationStructSwap`, we additionally localize the corrupted structural relation (edge) and report edge-level F1 (Rel-F1).

T4: Structural Propagation Tracing (Fusion360-Audit). For `RelationStructSwap`, we recover a dependency trace connecting the corrupted relation to the failure step. We define a reference trace as the type-prioritized shortest path on the assembly/dependency graph between the corrupted relation endpoints and the nearest text-referenced entities. We report trace edge-F1, normalized edit distance (NED; lower is better), and coverage, defined as the fraction of violated endpoints or variables covered by the trace.

In addition, we evaluate explanation compactness using the slice ratio $|\mathcal{U}|/T$, core size $|\mathcal{K}|$, and the average number of returned traces. For stochastic baselines, we assess determinism by running $N = 10$ repeated evaluations with identical inputs and reporting output consistency and standard deviation.

4.3 Baselines

We compare against unimodal, multimodal fusion, LLM-based auditing, and structure-only checks.

TextOnly: RoBERTa-base encoder + MLP classifier.

ImageOnly: ViT-B/16 encoder [32] + MLP classifier.

StructOnly (Fusion360-Audit): 3-layer GraphSAGE + MLP classifier.

EarlyFuse: concatenation of modality embeddings (text+image(+graph)) + MLP.

CrossAttn: cross-attention fusion across modalities.

TriFuse (Fusion360-Audit): gated tri-modal fusion with shared latent.

CLIPMatch (MyFixit-Audit): step-level text-image similarity scoring with a threshold and swap detection via bipartite matching.

GraphOnly (Fusion360-Audit): checks structural consistency on the graph (reachability/relation constraints) without executable state transitions or satisfiability solving.

LLM-CoT: LLM grader with constrained JSON output (decision, type, step, element); base model: GPT-4o mini.

LLM-Vote: self-consistency voting with $M = 7$ samples, aggregating decision/type/localization; base model: GPT-4o mini.

4.4 Implementation Details

Encoders and training. Unless otherwise stated, learned baselines use RoBERTa-base for text and ViT-B/16 [32] for images, while Fusion360 structure uses GraphSAGE with a hidden size of 256. We train all learned baselines with AdamW (lr = 1×10^{-4}, wd = 1×10^{-2}), a batch size of 64, and a maximum of 20 epochs, with early stopping based on validation macro-F1.

Our auditing framework. We parse each sample into a procedural step sequence. Each step is mapped to an action $a_t(\theta_t)$ with read/write sets and constraint templates (Section 3). Coupling constraints capture cross-step entity binding and count/spec consistency, while Fusion360 additionally defines relation constraints on assembly edges and mutual-exclusion rules. We maintain incremental constraint satisfiability with a per-instance timeout of 3 s. For conflict explanation, we use solver UNSAT cores with selector variables when supported; otherwise, we apply deterministic deletion-based shrinking on the sliced constraint set. Traces are extracted as type-prioritized shortest paths on the sliced dependency graph.

4.5 Main Results

Tables 3 and 4 report the main results. Overall, the proposed method improves (i) binary auditing accuracy, (ii) type discrimination under cross-modal perturbations, and (iii) localization and tracing quality, with the most significant gains on Count/Spec Perturbation and Relation/Structure Swap, where purely semantic fusion baselines are brittle. The improvements are consistent across both datasets, indicating that explicit dependency and constraint modeling generalizes beyond a single domain. In addition, the gains are not limited to accuracy: the framework provides more reliable tracing and localization, which is critical for actionable debugging.

Table 3: MyFixit-Audit results on the test set (3750 samples). T2 is a 5-way classification task (including Consistent). T3 is evaluated on inconsistent samples only (1875).

Method	T1: Binary		T2	T3: Step Localization				T3
	Acc↑	F1↑	mF1↑	Top-1↑	Top-3↑	Top-5↑	MRR↑	ElemAcc↑
TextOnly	0.812	0.810	0.642	0.311	0.512	0.621	0.431	0.336
ImageOnly	0.784	0.781	0.598	0.284	0.476	0.590	0.398	0.312
EarlyFuse	0.862	0.860	0.721	0.492	0.731	0.832	0.621	0.548
CrossAttn	0.903	0.902	0.781	0.603	0.842	0.915	0.731	0.672
CLIPMatch	0.886	0.884	0.752	0.571	0.821	0.904	0.708	0.651
LLM-CoT	0.892	0.890	0.771	0.566	0.803	0.892	0.695	0.636
LLM-Vote	0.914	0.913	0.802	0.621	0.858	0.928	0.742	0.689
Ours (Full)	0.953	0.953	0.872	0.742	0.921	0.962	0.835	0.786

Table 4: Fusion360-Audit results on the test set (1800 samples). T3/T4 are evaluated on inconsistent samples (900). T4 is evaluated on Relation/Structure Swap samples (180).

Method	T1: Binary		T2	T3: Step Localization				T3	T4: Trace (Rel/Struct)		
	Acc↑	F1↑	mF1↑	Top-1↑	Top-3↑	Top-5↑	MRR↑	Rel-F1↑	Edge-F1↑	NED↓	Cov↑
TextOnly	0.823	0.822	0.651	0.318	0.521	0.636	0.442	0.401	0.312	0.418	0.621
ImageOnly	0.791	0.789	0.612	0.296	0.487	0.602	0.416	0.382	0.287	0.443	0.603
StructOnly	0.846	0.845	0.683	0.402	0.642	0.751	0.536	0.558	0.441	0.332	0.701
EarlyFuse	0.884	0.883	0.734	0.521	0.771	0.863	0.648	0.621	0.512	0.284	0.744
CrossAttn	0.906	0.905	0.761	0.571	0.812	0.896	0.702	0.664	0.541	0.261	0.772
TriFuse	0.921	0.920	0.792	0.604	0.841	0.912	0.731	0.712	0.586	0.232	0.801
GraphOnly	0.862	0.861	0.701	0.553	0.784	0.871	0.681	0.641	0.503	0.298	0.762
LLM-CoT	0.901	0.900	0.752	0.571	0.809	0.892	0.701	0.663	0.522	0.273	0.774
LLM-Vote	0.914	0.913	0.781	0.602	0.836	0.907	0.728	0.671	0.541	0.254	0.786
Ours (Full)	0.951	0.951	0.854	0.731	0.902	0.947	0.822	0.804	0.742	0.176	0.873

4.6 Per-Type Performance

To analyze failure modes, Table 5 reports the per-type F1 scores for inconsistency classification (excluding the Consistent class) on both datasets, with results for MyFixit-Audit presented in the left panel and Fusion360-Audit in the right panel. The largest improvements are observed on Count/Spec Perturbation, which requires fine-grained numeric and specification consistency, and Relation/Structure Swap, which demands stronger structural reasoning capabilities.

Table 5: Per-type F1 for T2 inconsistency classification.

MyFixit-Audit					Fusion360-Audit				
Method	ImgSwap	Ent/Text	Cnt/Spec	RelSwap	Method	ImgSwap	Ent/Text	Cnt/Spec	Rel/Struct
CrossAttn	0.842	0.801	0.692	0.741	TriFuse	0.803	0.792	0.741	0.712
LLM-Vote	0.861	0.823	0.711	0.758	LLM-Vote	0.784	0.801	0.721	0.684
Ours (Full)	0.914	0.872	0.821	0.864	Ours (Full)	0.862	0.851	0.812	0.804

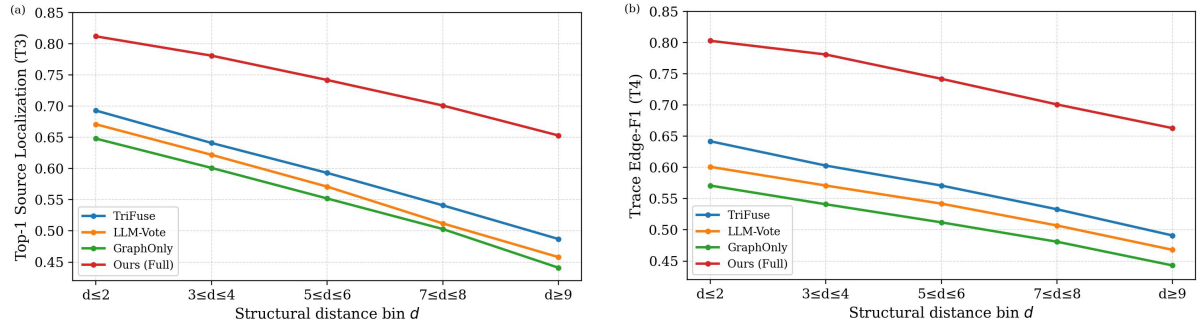
We ablate key components to quantify their impact on tri-modal auditing and structural tracing: (i) w/o dependency graph, (ii) w/o executable state transitions (structure-only), (iii) w/o coupling constraints, (iv) w/o mutex/structure constraints, (v) w/o provenance, (vi) w/o slicing, and (vii) w/o UNSAT core (naive first-failure attribution). Table 6 presents the complete ablation results on Fusion360-Audit. The dependency graph and executable state transitions are the most critical components, with their removal causing the largest performance drops across all tasks. Dependency slicing and UNSAT core extraction are essential for generating compact explanations, as their removal significantly increases both the slice ratio and core size. All other components contribute measurable improvements to classification and reasoning accuracy.

Table 6: Ablation results on Fusion360-Audit (test set).

Variant	T1		T2	T3			T4		Explain.	
	Acc↑	F1↑	mF1↑	Top-1↑	MRR↑	Rel-F1↑	Edge-F1↑	NED↓	Slice↓	Core↓
Ours (Full)	0.951	0.951	0.854	0.731	0.822	0.804	0.742	0.176	0.23	6.8
w/o dep. graph	0.928	0.928	0.812	0.603	0.712	0.701	0.561	0.264	0.41	7.9
w/o state	0.911	0.910	0.786	0.581	0.691	0.672	0.532	0.287	0.36	7.6
w/o coupling	0.936	0.936	0.823	0.651	0.761	0.732	0.621	0.222	0.25	7.1
w/o mutex	0.939	0.939	0.831	0.669	0.777	0.741	0.648	0.211	0.25	7.0
w/o prov.	0.949	0.949	0.852	0.681	0.789	0.791	0.663	0.214	0.24	6.9
w/o slicing	0.951	0.951	0.853	0.729	0.820	0.803	0.739	0.179	0.78	6.8
w/o UNSAT core	0.951	0.951	0.854	0.662	0.771	0.756	0.612	0.236	0.23	13.4

4.7 Long-Range Dependency Stress Test

We stress-test long-range propagation on Fusion360-Audit by stratifying Relation/Structure Swap samples using the structural distance d between the corrupted relation and the nearest entity mentioned in text, measured on the assembly graph. Fig. 2 shows that performance degrades as d increases for all methods: Fig. 2a demonstrates the drop in Top-1 localization accuracy, while Fig. 2b illustrates the decline in trace edge-F1. Despite this trend, the proposed verifier exhibits a notably flatter degradation curve compared to other baselines, maintaining substantially higher localization and tracing quality due to provenance-guided slicing and graph-grounded tracing.



(a) Top-1 accuracy for error source localization (T3). **(b)** Trace edge-F1 for structural propagation tracing (T4).

Figure 2: Long-range stress test on Fusion360-Audit. Performance degradation across different structural distance bins (d).

This trend should not be interpreted as a violation of the formal guarantee under the assumption of oracle structured inputs. Larger structural distances increase the number of grounded entities, intermediate dependencies, and candidate propagation paths. As a result, localization and trace matching become more sensitive to upstream grounding noise and to ambiguity in the benchmark reference trace, especially when multiple graph paths or conflict sources are logically plausible. Thus, the degradation reflects the difficulty of empirical localization and reference-trace matching in parsed multimodal data, not a loss of deterministic formal consistency checking once the structured audit instance is fixed.

4.8 Efficiency and Scalability

We measure the end-to-end auditing time per sample and provide a breakdown into three components: (i) incremental solving, (ii) dependency slicing, and (iii) core extraction plus path search. Table 7 presents the complete runtime statistics, with the median per-sample runtime breakdown shown in the left panel and the scalability analysis grouped by procedure length T in the right panel. Incremental maintenance substantially reduces computational cost compared with re-solving from scratch, and the generated explanations remain compact (slice ratio ≈ 0.23 and core size ≈ 6–7) even for longer procedures.

Table 7: Runtime statistics on Fusion360-Audit (median).

Per-Sample Runtime					**Runtime vs. Procedure Length**				
Component	Solve	Slice	Core+Path	Total	T bin	≤10	11–20	21–30	>30
Ours (Full)	86 ms	14 ms	23 ms	123 ms	Ours (Full)	71 ms	109 ms	153 ms	218 ms

4.9 Robustness to Upstream Noise

Although our focus is auditing rather than perception, upstream extraction may introduce noise. We simulate (i) step insertion/deletion, (ii) local step reordering within a window of size 3, and (iii) entity/parameter perturbations, with noise rate $\eta \in \{0, 5, 10, 20\}\%$. Table 8 shows that the proposed verifier degrades gracefully, maintaining strong formal consistency auditing and localization under moderate noise.

Table 8: Robustness on Fusion360-Audit under upstream noise.

Method	**T1 Acc↑**				**T3 Top-1↑**			
	0%	**5%**	**10%**	**20%**	**0%**	**5%**	**10%**	**20%**
TriFuse	0.921	0.908	0.892	0.861	0.604	0.583	0.561	0.517
LLM-Vote	0.914	0.901	0.883	0.852	0.602	0.571	0.542	0.498
GraphOnly	0.862	0.851	0.836	0.811	0.553	0.531	0.507	0.462
Ours (Full)	0.951	0.941	0.928	0.904	0.731	0.712	0.689	0.641

4.10 Determinism

Table 9 reports output consistency across repeated runs with identical inputs. LLM-based methods show non-zero variance in both accuracy and localization, while our verifier achieves perfect consistency (zero standard deviation) due to its fully deterministic execution and explanation pipeline.

Table 9: Determinism (Fusion360-Audit test set; $N = 10$ repeats).

Method	Consistency↑	σ(Acc)↓	σ(Top-1)↓
LLM-CoT	0.79	0.018	0.033
LLM-Vote	0.87	0.012	0.026
Ours (Full)	1.00	0.000	0.000

5 Conclusion

We presented a verification-centric framework for deterministic workflow auditing that couples typed dependency graphs with executable state transitions and uses slicing and conflict-core extraction to yield reproducible explanations. Across MyFixit-Audit and Fusion360-Audit, the approach delivers strong auditing accuracy and precise localization while maintaining compact explanations and low runtime, supporting practical debugging in multimodal procedural settings. Key contributions include (i) a dependency–state coupled verifier that enforces executability under constraints, (ii) provenance-guided slicing with UNSAT core based conflict identification, and (iii) deterministic, traceable explanations for long-range error propagation. Limitations include reliance on accurate upstream parsing and fixed action semantics, and the current focus on deterministic procedural workflows. We also find that compact conflict cores and graph-grounded traces make errors easier to interpret without sacrificing audit accuracy. These properties make the system a practical drop-in auditor for large-scale procedural datasets where reproducibility is critical. Future work will extend to richer stochastic settings, improve robustness to perception noise, and integrate learned parsers with formal guarantees.

Acknowledgement: Not applicable.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Xingxin Li; methodology, Jixin Xu, Xingxin Li; software, Jixin Xu; validation, Jixin Xu; formal analysis, Senlin Zhu; investigation, Qingqing Song; resources, Xingxin Li; data curation, Senlin Zhu; writing—original draft preparation, Jixin Xu; writing—review and editing, Qingqing Song, Jixin Xu; visualization, Jixin Xu; supervision, Xingxin Li; project administration, Xingxin Li; funding acquisition, Xingxin Li. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The source code, schema/template definitions, perturbation scripts, split files, configuration files, and evaluation scripts will be released in an anonymized repository upon acceptance. Due to licensing restrictions of the original repair manuals and CAD assets, raw data may not be fully redistributed; however, processed annotations, split identifiers, and scripts for reproducing all reported tables will be provided.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Dalvi B, Huang L, Tandon N, Yih WT, Clark P. Tracking state changes in procedural text: a challenge dataset and models for process paragraph comprehension. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Stroudsburg, PA, USA: ACL; 2018. p. 1595–604. doi:10.18653/v1/N18-1144.

2. Zhukov D, Alayrac JB, Cinbis RG, Fouhey D, Laptev I, Sivic J. Cross-task weakly supervised learning from instructional videos. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Piscataway, NJ, USA: IEEE; 2019. p. 3537–45. doi:10.1109/CVPR.2019.00365.
3. Shridhar M, Thomason J, Gordon D, Bisk Y, Han W, Mottaghi R, et al. ALFRED: a benchmark for interpreting grounded instructions for everyday tasks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Piscataway, NJ, USA: IEEE; 2020. p. 10740–9. doi:10.1109/CVPR42600.2020.01075.
4. Seff A, Ovadia Y, Zhou W, Adams RP. SketchGraphs: a large-scale dataset for modeling relational geometry in computer-aided design. arXiv:2007.08506. 2020.
5. Willis KDD, Pu Y, Luo J, Chu H, Du T, Lambourne JG, et al. Fusion 360 gallery: a dataset and environment for programmatic CAD construction from human design sequences. ACM Trans Graph. 2021;40(4):54. doi:10.1145/3450626.3459818.
6. Wei J, Wang X, Schuurmans D, Bosma M, Ichter B, Xia F, et al. Chain-of-thought prompting elicits reasoning in large language models. arXiv:2201.11903. 2022.
7. Wang X, Wei J, Schuurmans D, Le QV, Chi EH, Narang S, et al. Self-consistency improves chain of thought reasoning in language models. arXiv:2203.11171. 2023.
8. Madaan A, Tandon N, Gupta P, Hallinan S, Gao L, Wiegreffe S, et al. Self-Refine: iterative refinement with self-feedback. arXiv:2303.17651. 2023.
9. van der Aalst WMP. Process mining: data science in action. 2nd ed. Berlin/Heidelberg, Germany: Springer; 2016. doi:10.1007/978-3-662-49851-4.
10. Rozinat A, van der Aalst WMP. Conformance checking of processes based on monitoring real behavior. Inf Syst. 2008;33(1):64–95. doi:10.1016/j.is.2007.07.001.
11. Adriansyah A, van Dongen BF, van der Aalst WMP. Conformance checking using cost-based fitness analysis. In: Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2011). Piscataway, NJ, USA: IEEE; 2011. p. 55–64. doi:10.1109/EDOC.2011.12.
12. Percassi F, Scala E, Vallati M. The power of reformulation: from validation to planning in PDDL+. Proc Int Conf Autom Plan Sched. 2022;32:288–96. doi:10.1609/icaps.v32i1.19812.
13. de Moura L, Bjørner N. Z3: an efficient SMT solver. In: Tools and algorithms for the construction and analysis of systems (TACAS 2008). Berlin/Heidelberg, Germany: Springer; 2008. p. 337–40. doi:10.1007/978-3-540-78800-3_24.
14. Ding ZJ, Li S, Chen C, He C. Program dependence net and on-demand slicing for property verification of concurrent system and software. J Syst Softw. 2025;219(2):112221. doi:10.1016/j.jss.2024.112221.
15. Agrawal H, Horgan JR. Dynamic program slicing. In: Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation (PLDI). New York, NY, USA: ACM; 1990. p. 246–56. doi:10.1145/93542.93576.
16. Buneman P, Khanna S, Tan WC. Why and where: a characterization of data provenance. In: Van den Bussche J, Vianu V, editors. Database theory—ICDT 2001. Berlin/Heidelberg, Germany: Springer; 2001. p. 316–30. doi:10.1007/3-540-44503-X_20.
17. Green TJ, Karvounarakis G, Tannen V. Provenance semirings. In: Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS). New York, NY, USA: ACM; 2007. p. 31–40. doi:10.1145/1265530.1265535.
18. Reiter R. A theory of diagnosis from first principles. Artif Intell. 1987;32(1):57–95. doi:10.1016/0004-3702(87)90062-2.
19. Liffiton MH, Sakallah KA. Algorithms for computing minimal unsatisfiable subsets of constraints. J Autom Reason. 2008;40(1):1–33. doi:10.1007/s10817-007-9084-z.
20. Murata T. Petri nets: properties, analysis and applications. Proc IEEE. 1989;77(4):541–80. doi:10.1109/5.24143.
21. Gocht S, Balyo T. Accelerating SAT based planning with incremental SAT solving. In: Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017). Palo Alto, CA, USA: AAAI Press; 2017. p. 135–9. doi:10.1609/icaps.v27i1.13798.

22. Sülflow A, Fey G, Bloem R, Drechsler R. Debugging design errors by using unsatisfiable cores. In: 11th ITG/GMM/GI-Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV). Aachen, Germany: Shaker Verlag; 2008. p. 159–68. doi:10.1142/S0129054107004942.
23. Miech A, Zhukov D, Alayrac JB, Tapaswi M, Laptev I, Sivic J. HowTo100M: learning a text-video embedding by watching hundred million narrated video clips. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV). Piscataway, NJ, USA: IEEE; 2019. p. 2630–40. doi:10.1109/ICCV.2019.00272.
24. Nabizadeh N, Kolossa D, Heckmann M. MyFixit: an annotated dataset, annotation tool, and baseline methods for information extraction from repair manuals. In: Proceedings of the Twelfth Language Resources and Evaluation Conference (LREC). Marseille, France: ELRA; 2020. p. 2120–8.
25. Willis KDD, Jayaraman PK, Chu H, Tian Y, Li Y, Grandi D, et al. JoinABLe: learning bottom-up assembly of parametric CAD joints. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Piscataway, NJ, USA: IEEE; 2022. p. 15828–39.
26. Bettig B, Hoffmann CM. Geometric constraint solving in parametric computer-aided design. J Comput Inf Sci Eng. 2011;11(2):021001. doi:10.1115/1.3593408.
27. Geng S, Cooper H, Moskal M, Jenkins S, Berman J, Ranchin N, et al. Generating structured outputs from language models: benchmark and studies. arXiv:2501.10868. 2025.
28. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. In: Advances in Neural Information Processing Systems 30 (NIPS 2017). Red Hook, NY, USA: Curran Associates, Inc.; 2017. p. 5998–6008.
29. Liu Y, Ott M, Goyal N, Du J, Joshi M, Chen D, et al. RoBERTa: a robustly optimized BERT pretraining approach. arXiv:1907.11692. 2019.
30. Hamilton WL, Ying R, Leskovec J. Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems 30 (NeurIPS 2017). Red Hook, NY, USA: Curran Associates, Inc.; 2017. p. 1024–34.
31. Radford A, Kim JW, Hallacy C, Ramesh A, Goh G, Agarwal S, et al. Learning transferable visual models from natural language supervision. In: Proceedings of the 38th International Conference on Machine Learning. Proceedings of Machine Learning Research. Cambridge, MA, USA: PMLR; 2021. p. 8748–63.
32. Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, et al. An image is worth 16x16 words: transformers for image recognition at scale. arXiv:2010.11929. 2021.