ScieRxiv
NOT PEER-REVIEWED
Original Article

# WikiPlanner: Wiki-derived temporal monitors for safe LLM Agent planning

**Bailing Zhang**

Institute of Data Science and Intelligent Technology, NingboTech University, Ningbo 315336, Zhejiang, China.

**Correspondence to:** Dr. Bailing Zhang, Institute of Data Science and Intelligent Technology, NingboTech University, Ningbo 315336, Zhejiang, China. E-mail: bailing.zhang1961@gmail.com

**Received: 07 June 2026 | Approved: 26 June 2026 | Online: 26 June 2026**

## Abstract

Large language model (LLM) agents in high-stakes sequential planning tasks may produce individual steps that appear locally reasonable while the overall sequence violates critical domain constraints. Existing single-step filtering approaches cannot detect such cross-step temporal ordering violations. This paper presents WikiPlanner, a framework that extracts temporal constraints from domain wikis, structures them via a typed Temporal Constraint Intermediate Representation (TC-IR), and compiles them into parallel runtime monitors. Each monitor directly implements the operational semantics of its corresponding Linear Temporal Logic over finite traces (LTLf) formula. The two-stage pipeline —TC-IR extraction followed by deterministic template compilation—isolates LLM extraction uncertainty from formal constraint compilation. Evaluation on a 45-constraint medical planning benchmark shows that WikiPlanner achieves $F1 = 0.920$ on a controlled aligned test set where violation plans are specifically constructed to match monitored constraints, and $F1 = 0.582$ on a natural plan set without such alignment, revealing action grounding as the primary deployment bottleneck. Ablation analysis confirms a super-additive interaction between wiki-

derived constraints and sequential monitoring (+0.132 F1 beyond additive prediction), and validation latency scales linearly with constraint count (0.007-0.063 ms). Template repair resolves violations in 39.1% of cases (average 0.56 edits), with insertion-type repairs substantially outperforming reordering repairs. The key finding is that compiling wiki knowledge into reusable temporal monitors is feasible and effective in controlled settings; action vocabulary alignment remains the main obstacle to natural-plan deployment.

**Keywords:** LLM agent planning, temporal logic, runtime verification, knowledge-based constraints, clinical decision support

## INTRODUCTION

Deploying LLM agents in consequential domains requires more than generating locally plausible actions: the complete action sequence must conform to domain-specific temporal ordering rules. A clinician agent that prescribes an ACE inhibitor may act on sound local reasoning, but failure to schedule renal function monitoring within the following visits violates a standard safety guideline that independent per-step checking cannot detect. Similar issues arise in industrial control, pharmaceutical management, and regulated service workflows, where the order and timing of actions carry safety implications that no single step expresses in isolation.

Recent work on constraining LLM outputs has focused on single-output filtering[1], prompt-based self-verification[2], and constitutional approaches[3]. Runtime verification in classical software engineering[4,5] checks execution traces against formal specifications, but has not been systematically adapted to settings where constraints derive from natural-language domain wikis and must be applied to LLM-generated action sequences. Closer to our problem, Agent-C[6] enforces temporal safety properties during LLM token generation using SMT solving, achieving perfect conformance on retail and airline reservation tasks; RvLLM[7] performs runtime verification of LLM outputs against domain-specific predicates encoded in a custom specification language. Both works assume that constraints are manually authored by domain experts or encoded as task specifications.

WikiPlanner addresses a complementary knowledge-engineering problem: how to

automatically derive temporal constraints from maintained domain wikis and compile them into reusable runtime monitors, without requiring manual specification authoring. Domain wikis encode rich temporal ordering knowledge —prerequisite relationships, mandatory follow-up requirements, time-bounded monitoring windows, mutual exclusions, and forbidden action sequences—that can serve as machine-enforceable constraints if compiled into an appropriate formal representation.

The core technical challenge is that direct LLM-to-formal-logic compilation is unreliable: LLMs confuse temporal direction, misspecify time bounds, and produce invalid formula syntax. WikiPlanner addresses this by introducing a Temporal Constraint Intermediate Representation (TC-IR), a typed JSON schema that captures constraint semantics before any formal compilation. LLMs target TC-IR extraction, and a deterministic template compiler then maps each record to an LTLf formula implemented as a parallel typed monitor.

This paper makes the following contributions:

1.We formulate the problem of wiki-derived temporal runtime monitoring for LLM agent plans, where constraints are automatically extracted from maintained domain wiki sources rather than manually specified.

2.We introduce TC-IR, a typed intermediate representation that isolates LLM extraction uncertainty from deterministic monitor compilation, supporting six semantically distinct temporal constraint classes.

3.We implement a family of parallel typed finite-trace monitors enabling source-localized violation reporting without product-automaton construction.

4.We provide both controlled and natural-plan evaluations on a medical planning benchmark, showing strong detection in aligned settings while identifying action grounding and schema expressivity as deployment bottlenecks.

5.We analyse template repair behaviour and show that local repairs work well for insertion-type violations but are insufficient for global ordering conflicts.

## RELATED WORK

### LLM planning and its limitations

Multiple studies document systematic failures of LLMs in classical planning tasks[2,8,17]. Kambhampati *et al.*[9] argue that LLMs are best used as heuristic generators within modular planning frameworks; Valmeekam *et al.*[2] show that LLMs frequently violate global constraints while generating locally plausible steps. LLM hallucinations[10] compound this problem in knowledge-intensive domains.

### Runtime monitoring and temporal logic

Runtime verification[4,5] checks execution traces against formal specifications. Classical LTL[11] targets infinite traces; De Giacomo and Vardi[12] introduced LTLf for finite traces, admitting DFA representations via tools such as ltlf2dfa[13]. Enforcing temporal constraints on generative agents using Temporal Stream Logic (TSL) has been explored by Rothkopf *et al.*[14], achieving over 96% adherence versus 14.67% for unconstrained LLMs on structured agent tasks. WikiPlanner adopts LTLf as its formal backbone but replaces general DFA compilation with typed monitors implementing the acceptance condition per constraint class, avoiding product-automaton state explosion.

### Temporal constraint enforcement for LLM agents

Agent-C[6] presents a framework that provides runtime guarantees for LLM agents by translating manually authored temporal properties into first-order logic and using SMT solving to enforce compliance during token generation, achieving 100% conformance on retail and airline service tasks. RvLLM[7] verifies LLM outputs against domain-specific constraints encoded in a custom specification language (ESL), targeting single-output violations in legal and numerical domains. Both require expert-authored constraint specifications.

WikiPlanner is complementary: rather than enforcing manually specified constraints, it addresses the knowledge engineering problem of automatically compiling constraints from domain wikis into reusable monitors. This distinction—wiki-sourced, automatically extracted, sequence-level—defines WikiPlanner's position relative to both Agent-C and RvLLM.

### Knowledge bases and Wiki-as-constraint

Retrieval-augmented generation (RAG)[3] augments LLMs with retrieved text at inference time, but retrieved passages are not enforced as hard constraints. Karpathy[15] argues that structured wiki layers maintained separately from the generative model provide verifiable, updatable knowledge for agent systems. WikiPlanner operationalises this view by treating wiki content as a compilable constraint specification rather than a retrieval source.

## THE WIKIPLANNER FRAMEWORK

### Problem setting

Let denote a finite action vocabulary and let a plan $\pi = (a_1, a_2, \ldots, a^n)$ be an ordered sequence of actions $a_i \in$ . A domain wiki is a collection of natural-language documents encoding temporal ordering knowledge about actions in . A temporal constraint is a tuple $c = (\tau, \alpha, \beta, k, \sigma)$, where $\tau$ is the constraint type, $\alpha$ the trigger action, $\beta$ the related action, $k$ an optional step bound, and $\sigma \in \{\text{hard}, \text{soft}\}$ the severity. WikiPlanner aims to: (i) detect whether $\pi$ violates any constraint derivable from , and (ii) repair violations by producing a minimally modified plan $\pi'$ satisfying all hard constraints.

An important scope boundary: WikiPlanner processes ordinal temporal constraints (action ordering and within-k-step requirements) rather than real-time metric constraints. Time-bounded requirements (e.g., “within 48 h”) are discretised to step counts under a one-step-per-clinical-contact assumption, which is appropriate for clinic-scale planning but would require extension to Metric Temporal Logic (MTL) for settings with variable inter-step durations.

### Framework overview

Figure 1 shows the four-stage pipeline. (1) GLM-4-flash reads chunked wiki text and outputs TC-IR records. (2) A deterministic template compiler maps each TC-IR record to an LTLf formula. (3) Each formula is instantiated as an independent typed monitor. (4) Incoming LLM plans are normalised to vocabulary symbols and validated against all monitors in parallel; violations trigger the repair module.

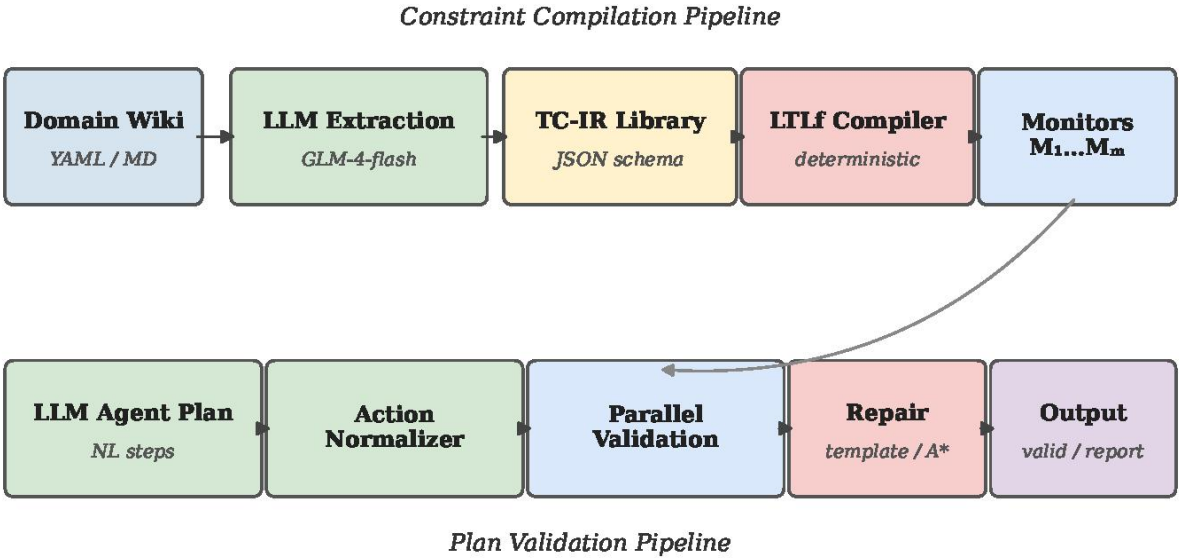


**Figure 1.** WikiPlanner four-stage pipeline. Wiki text is parsed by GLM-4-flash into TC-IR records, compiled to LTLf formulas by a deterministic template compiler, and instantiated as parallel typed monitors. Incoming LLM plans are normalised to vocabulary symbols and validated; violations trigger the repair module.

### TC-IR schema and constraint classes

The Temporal Constraint Intermediate Representation (TC-IR) is a typed JSON record separating LLM-based extraction from deterministic LTLf compilation. Each record contains: constraint_type (one of six classes), trigger_action (snake_case identifier), related_actions (type-dependent secondary actions), time_bound_steps (optional integer), severity (hard/soft), and confidence (LLM-assigned, in [0,1]).

Table 1 defines the six constraint classes. We consolidate RESPONSE into MANDATORY_FOLLOWUP at the LTLf formula level (both express $G(\alpha \rightarrow F\beta)$), while retaining the distinction in TC-IR for clinical annotation: RESPONSE marks emergency or safety-triggered requirements (e.g., anaphylaxis → epinephrine) while MANDATORY_FOLLOWUP marks protocol-level follow-up (e.g., ACEI → eGFR check). Both enforce the same temporal logic but differ in severity and clinical urgency, which downstream triage systems may exploit.

A correction from an earlier draft: the MUTUAL_EXCLUSION LTLf template was incorrectly stated as $G(\neg(\alpha \wedge \beta))$. Since each plan step has a single action, $\alpha$ and $\beta$ cannot be simultaneously true at any step, making that formula vacuously satisfied and

useless. The correct formula is $\neg(F\alpha \wedge F\beta)$, which rejects any trace containing both α and β at any positions.

**Table 1. TC-IR constraint classes, operational semantics, and LTLf formula templates. α = trigger, β = related action, k = step bound, γ = guard action.**

| Class | Operational semantics | LTLf formula |
|---|---|---|
| PREREQUISITE | Reject at α if β has not been observed before | $\neg\alpha \, W \, \beta$ |
| MANDATORY_FOLLOWUP | Reject at end of trace if no β occurs after α | $G(\alpha \rightarrow F\beta)$ |
| BOUNDED_FOLLOWUP | Reject if β does not appear within k steps after α | $G(\alpha \rightarrow \vee_{i=0}^{k} X^i\beta)$ |
| MUTUAL_EXCLUSION | Reject if both α and β appear anywhere in trace | $\neg(F\alpha \wedge F\beta)$ |
| FORBIDDEN_SEQUENCE | After α, reject if β appears before guard γ | $G(\alpha \rightarrow \neg(\neg\gamma \, U \, \beta))$ |
| ORDERING | Reject at β if α has not occurred | $\neg\beta \, W \, \alpha$ |
| RESPONSE* | Same as MANDATORY_FOLLOWUP; annotated for emergency/safety triggers | $G(\alpha \rightarrow F\beta)$ |

**RESPONSE and MANDATORY_FOLLOWUP share the same LTLf formula; they differ in TC-IR severity and clinical semantics.*

**Action normalization**

LLM-generated plans use natural-language step descriptions that may diverge from TC-IR vocabulary identifiers. A two-pass normalizer maps each step to the nearest vocabulary symbol: (1) exact synonym matching against a 42-symbol domain vocabulary; (2) GLM-4-flash semantic alignment for unmatched steps. Steps that cannot be resolved are flagged as UNKNOWN and excluded from monitor evaluation. In our experiments, the average proportion of unknown actions per plan was 0.20 on the

controlled vocabulary-constrained plans; this rises substantially on natural plans, constituting the primary gap between aligned-set and natural-set performance.

### Parallel typed monitors

Given a TC-IR record set $C = \{c_1,\ldots,c_m\}$, each record instantiates one typed monitor. All monitors run in parallel over the plan; the validator collects every violation rather than stopping at the first, enabling source-localised violation reporting.

Definition 1 (Typed Monitor). A typed monitor $M_i$ for constraint $c_i = (\tau, \alpha, \beta, k, \sigma)$ is a function $M_i$: $* \rightarrow \{0,1\} \times \mathbb{Z}_{\geq -1}$ that directly implements the acceptance condition for the LTLf formula $\phi_i$ associated with $c_i$. In the codomain, $\mathbb{Z}_{\geq -1}$ denotes the set of integers $\geq -1$: the value $-1$ is a sentinel for "no violation"; non-negative values are the zero-based index of the first violating step.

Each monitor runs in O(n) time over the plan ($n$ = plan length). Total validation cost is O(mn) where $m = |C|$. The sum of DFA state counts $\Sigma|Q_i|$ across all monitors is substantially smaller than the product $\Pi|Q_i|$ that a single merged DFA would require, avoiding state-space explosion.

### Two-stage repair

Upon detecting hard violations, WikiPlanner attempts two-stage repair. Stage 1 (template repair) applies a type-specific minimal edit: PREREQUISITE violations insert the missing prior action before the trigger; MANDATORY_FOLLOWUP and RESPONSE insert the missing follow-up after the trigger; BOUNDED_FOLLOWUP moves the follow-up action within the required window; MUTUAL_EXCLUSION removes the excluded action; FORBIDDEN_SEQUENCE inserts a guard action between trigger and forbidden action; ORDERING transposes the two actions. The repaired plan is re-validated immediately. Stage 2 (A* search) is included as a preliminary fallback for residual violations, bounded to five single-step edits over the domain vocabulary; it is evaluated exploratorily rather than as a primary contribution.

## RESULTS AND DISCUSSION

### Experimental setup

Experiments use a medical planning domain built from two wiki repositories: a GP

clinic wiki (10 YAML case files) and a medical school wiki (10 YAML case files) covering chronic disease management, psychiatry, emergency medicine, and pharmacology[18]. GLM-4-flash[16] extracted TC-IR records from chunked wiki text; confidence filtering (threshold = 0.60) and duplicate removal yielded 37 auto-extracted constraints. Eight additional constraints covering BOUNDED_FOLLOWUP, MUTUAL_EXCLUSION, and RESPONSE—under-represented by auto-extraction—were manually authored and human-verified. The final library contains 45 constraints across all seven TC-IR classes. Zero LTLf compilation errors were observed.

Two test sets were constructed. The aligned set (137 plans: 50 correct, 87 violation) uses constraint-specific violation directives to generate each violation plan (e.g., "include prescribe_acei but not check_egfr"), ensuring ground-truth alignment with monitored constraints. This set validates monitor correctness but overstates real-world performance. The natural set (99 plans: 50 correct, 49 violation) generates plans without constraint-specific directives, modelling realistic deployment where the generator does not know which action pairs are monitored. All plans use a 42-symbol action vocabulary; plans on the natural set may use synonyms outside this vocabulary.

Baselines: B0 (no validation), B1 (per-action danger list), B2 (bag-of-words action presence rules), B3 (10 handcrafted sequential rules), B5 (WikiPlanner-Lite: TC-IR with action presence checking only). Metrics: precision, recall, F1, false rejection rate (FRR = FP/(FP + TN)), and per-plan latency. All validation is CPU-bound on an Intel-based workstation.

**Main results**

Table 2 reports results on the aligned test set. WikiPlanner-Full achieves F1 = 0.920, improving by 0.268 over the best handcrafted sequential baseline (B3, F1 = 0.652) and by 0.224 over the best ordering-agnostic baseline (WikiPlanner-Lite, F1 = 0.696). Equal precision and recall (0.920) indicate calibrated coverage across violation types. Validation latency is 0.1 ms per plan.

On the natural set, WikiPlanner-Full achieves F1 = 0.582 (precision = 0.767, recall = 0.469, FRR = 0.140). The 0.338 gap between the two sets is attributable to vocabulary misalignment: when the plan generator uses synonyms outside the 42-

symbol vocabulary (e.g., prescribe_drugs instead of prescribe_acei), the normaliser cannot map the step to a monitored symbol, and the corresponding constraint does not fire. LLM self-checking (B4, GLM-4-flash) achieved F1 = 0.429 on the natural set with average latency 1,423.5 ms, substantially slower and weaker.

**Table 2. Results on the aligned test set (137 plans). WikiPlanner-Full uses all 45 constraints.**

| Method | Precision | Recall | F1 | FRR | Latency (ms) |
|---|---|---|---|---|---|
| B0 No Validation | 0.000 | 0.000 | 0.000 | 0.000 | 0.0 |
| B1 Single-step | 0.857 | 0.138 | 0.238 | 0.040 | 0.0 |
| B2 Bag-of-Words | 0.921 | 0.402 | 0.560 | 0.060 | 0.0 |
| B3 Sequential Rules (10) | 0.852 | 0.529 | 0.652 | 0.160 | 0.0 |
| B5 WikiPlanner-Lite | 0.941 | 0.552 | 0.696 | 0.060 | 0.0 |
| WikiPlanner-Verified (8) | 0.860 | 0.494 | 0.628 | 0.140 | 0.1 |
| WikiPlanner-Full (45) | 0.920 | 0.920 | 0.920 | 0.140 | 0.1 |

**Table 3. Per-type violation detection rates for WikiPlanner-Full on the aligned test set.**

| Constraint type | Detected / Total | Detection rate |
|---|---|---|
| BOUNDED_FOLLOWUP | 8/8 | 100% |
| MUTUAL_EXCLUSION | 8/8 | 100% |
| RESPONSE | 8/8 | 100% |
| PREREQUISITE | 19/20 | 95% |
| ORDERING | 18/20 | 90% |
| MANDATORY_FOLLOWUP | 14/16 | 88% |
| FORBIDDEN_SEQUENCE | 5/7 | 71% |
| Overall | 80/87 | 92% |

## Ablation study

*A1: Component ablation*

Table 4 decomposes F1 contributions by constraint source (handcrafted vs. wiki-derived) and checking mode (bag-of-words vs. sequential). Sequential checking alone adds +0.092 F1 over handcrafted rules and +0.224 F1 over wiki-derived rules. The interaction between wiki-derived constraints and sequential checking exceeds the additive prediction by +0.132 F1. This super-additive effect arises because wiki-derived constraints cover constraint types (e.g., BOUNDED_FOLLOWUP) that are invisible to bag-of-words checking but fully exploited by sequential monitors.

**Table 4. 2 × 2 component ablation (F1 scores).**

| | **Bag-of-Words** | **Sequential** | **Δ (seq − BoW)** |
|---|---|---|---|
| Handcrafted constraints | 0.560 | 0.652 | +0.092 |
| Wiki-IR constraints | 0.696 | 0.920 | +0.224 |
| Δ (wiki − craft) | +0.136 | +0.268 | |

*A2: Constraint source*

Auto-extracted constraints alone ($n = 37$) achieve F1 = 0.766 with FRR = 0.000, meaning they never falsely reject correct plans. Human-verified constraints alone ($n = 8$) yield F1 = 0.628 with FRR = 0.140. The combined library achieves F1 = 0.920 but inherits FRR = 0.140 from the verified constraints. This counter-intuitive result is analysed in the failure mode discussion below.

*A3: Scalability*

Validation latency scales linearly with the number of active monitors: 0.007 ms (5 constraints) to 0.063 ms (45 constraints). F1 increases monotonically from 0.621 to 0.920 across the same range (5, 10, 15, 20, 30, 45 constraints), confirming that adding constraints increases coverage without introducing false positives from auto-extracted constraints. Figure 2 plots both metrics as functions of constraint count: latency is linear (slope ≈ 0.0013 ms per constraint, $R^2 > 0.999$), and F1 is monotonically increasing from 0.621 to 0.920.

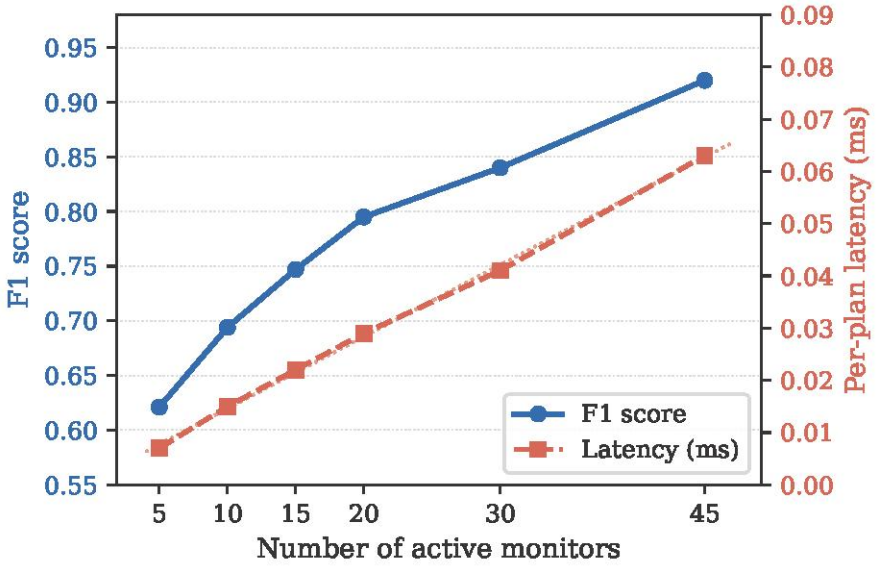


**Figure 2.** F1 score (left axis, filled circles) and per-plan validation latency in milliseconds (right axis, open squares) as functions of the number of active monitors. The dashed line is a linear regression fit to latency ($R^2 > 0.999$). Constraint subsets are formed by adding constraints in descending confidence order.

*A4: Template Repair*

Table 5 reports repair success rates by constraint type. The overall success rate is 39.1% (34/87), below the 75% design target. Insertion-type repairs (MANDATORY_FOLLOWUP: 75%; BOUNDED_FOLLOWUP: 62.5%) succeed at substantially higher rates than reordering repairs (ORDERING: 20%; PREREQUISITE: 20%). Transposing actions to fix ordering violations frequently introduces new violations with different constraints, compounding the failure. The current single-pass implementation also cannot address multi-violation plans where fixing one violation exposes another. Template repair is best characterised as an exploratory diagnostic module effective for insertion-type violations; iterative repair and constraint-aware search are left as future work.

**Table 5. Template repair success rate by constraint type.**

| Constraint type | Success/Total | Rate | Avg. edits |
|---|---|---|---|
| MANDATORY_FOLLOWUP | 12/16 | 75.0% | 0.42 |

| Constraint type | Success/Total | Rate | Avg. edits |
|---|---|---|---|
| BOUNDED_FOLLOWUP | 5/8 | 62.5% | 0.60 |
| MUTUAL_EXCLUSION | 4/8 | 50.0% | 0.50 |
| FORBIDDEN_SEQUENCE | 3/7 | 42.9% | 0.67 |
| RESPONSE | 2/8 | 25.0% | 0.50 |
| ORDERING | 4/20 | 20.0% | 0.75 |
| PREREQUISITE | 4/20 | 20.0% | 0.75 |
| Overall | 34/87 | 39.1% | 0.56 |

Figure 3 visualises the per-type repair success rates and the distribution of repair methods. The left panel highlights the dichotomy between insertion-type repairs (MANDATORY_FOLLOWUP, BOUNDED_FOLLOWUP) with high success rates and reordering repairs (ORDERING, PREREQUISITE) with low success rates. The right panel shows that move_to_window and insert_prerequisite are the most frequently applied methods, together accounting for over half of all repair attempts; nine cases match no template.

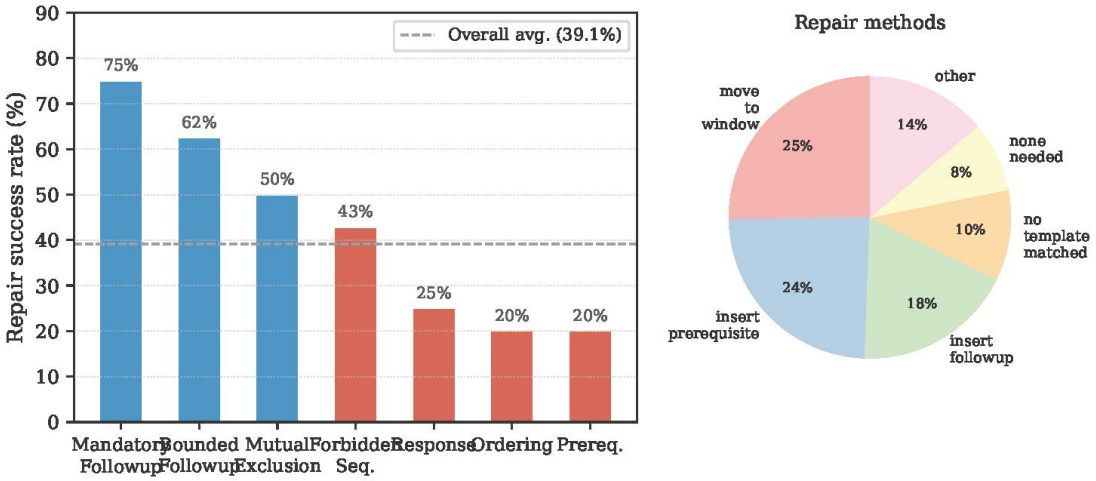


**Figure 3.** Left panel: template repair success rate by constraint type. Blue bars indicate ≥ 50% success; red bars indicate < 50% success. Dashed line marks the overall average (39.1%). Right panel: distribution of repair methods applied across all 87 violation plans.

**Failure mode analysis**

Four classes of failure characterise WikiPlanner's current limitations.

Vocabulary alignment failure. When a plan generator uses synonyms outside the monitored vocabulary (e.g., prescribe_drugs instead of prescribe_acei), the normaliser fails to map the step, and the constraint does not fire. This accounts for the majority of the 0.338 F1 gap between aligned and natural sets. Resolving this requires tighter co-design between the generator and the action vocabulary, or a more robust semantic normaliser.

Pre-emptive monitoring and schema insufficiency. Seven correct plans are falsely rejected because they perform a required check before prescribing (e.g., check_egfr → prescribe_acei → schedule_followup), while the BOUNDED_FOLLOWUP monitor requires the check to appear after the trigger. The current TC-IR schema cannot express bidirectional temporal windows ("before or after"). Notably, auto-extracted constraints produce FRR = 0.000; all false rejections originate from the eight human-verified BOUNDED_FOLLOWUP and RESPONSE constraints, whose strict post-trigger windows conflict with pre-emptive clinical practice patterns.

Temporal granularity mismatch. The discretisation of clock-hour windows to step counts is adequate for clinic-scale planning under the one-step-per-contact assumption, but inappropriate for settings with variable inter-step durations. Extension to Metric Temporal Logic would remove this assumption at increased computational cost.

Constraint provenance and authority. Domain wikis may be incomplete, outdated, or context-dependent. The current TC-IR schema does not capture constraint provenance (source document, evidence span, version, expert verification status). Without provenance, evaluators cannot assess the authority of wiki-derived constraints or audit violation reports. Adding source metadata and verification status fields is a concrete near-term improvement.

**Limitations**

Three limitations deserve explicit statement. First, the aligned-set evaluation (F1 = 0.920) uses violation plans generated with constraint-specific directives that align

with monitored constraints; this validates monitor implementation correctness but overstates deployment performance relative to the natural set (F1 = 0.582). Second, the extraction quality of the TC-IR pipeline (precision, recall, and type accuracy of constraint extraction from wiki text) was not formally evaluated against human-annotated ground truth; this is an important gap for future work. Third, experiments cover only one domain (medical planning) at pilot scale (137 plans); results should not be extrapolated to other domains without separate validation.

## CONCLUSIONS

WikiPlanner demonstrates that temporal constraints can be automatically extracted from domain wikis and compiled into parallel runtime monitors for validating LLM agent plans. The two-stage TC-IR pipeline isolates LLM extraction uncertainty from deterministic compilation, enabling transparent, source-localised violation reporting at sub-millisecond latency. On a controlled medical planning benchmark, WikiPlanner-Full achieves F1 = 0.920 with linear latency scaling, and exhibits a super-additive interaction between wiki-derived constraints and sequential monitoring (+ 0.132 beyond additive). On natural plans without vocabulary alignment, performance drops to F1 = 0.582, identifying action grounding as the primary deployment challenge.

The paper also corrects a formal error in an earlier version of the MutualExclusion LTLf template and consolidates the Response and MandatoryFollowup classes at the formula level while preserving their clinical semantic distinction in TC-IR. Four failure mode classes are identified: vocabulary alignment, schema insufficiency (no bidirectional temporal windows), temporal granularity mismatch, and missing constraint provenance. These constitute a concrete research agenda for future iterations of the wiki-to-monitor pipeline.

Future directions include: formal evaluation of TC-IR extraction quality against human-annotated wiki corpora; extension of the TC-IR schema to support bidirectional temporal windows and exception conditions; robust semantic action normalisation using embedding retrieval; multi-domain evaluation including industrial process control; and iterative multi-pass repair for ordering violations.

## DECLARATIONS

## Authors' contributions

The author contributed solely to the article.

## Availability of data and materials

Code and data will be released at https://github.com/aussie-bzhang/wikiplanner upon acceptance.

## Financial support and sponsorship

None.

## Conflicts of interest

All authors declared that there are no conflicts of interest.

## Ethical approval and consent to participate

Not applicable. No human participants, human material, or identifiable human data were used. Wiki sources are de-identified educational documents.

## Consent for publication

Not applicable.



## REFERENCES

[1] Bai Y, Jones S, Ndousse K, et al. Constitutional AI: Harmlessness from AI feedback. *arXiv:*2212.08073; 2022.[DOI: 10.48550/arXiv.2212.08073]

[2] Valmeekam K, Marquez M, Sreedharan S, Kambhampati S. On the planning abilities of large language models—a critical investigation. *Adv Neural Inf Process Syst.* 2023;36.[DOI: 10.48550/arXiv.2302.06706]

[3] Lewis P, Perez E, Piktus A, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Adv Neural Inf Process Syst.* 2020;33:9459-74.[DOI: 10.48550/arXiv.2005.11401]

[4] Leucker M, Schallhart C. A brief account of runtime verification. *J Log Algebr Methods Program.* 2009;78(5):293-303.[DOI: 10.1016/j.jlap.2008.08.004]

[5] Havelund K, Rosu G. Synthesizing monitors for safety properties. *In: Proc.* TACAS 2002. LNCS 2280; Springer; 2002. p. 342-56.[DOI: 10.1007/3-540-46002-0_24]

[6] Kamath A, Zhang S, Xu C, et al. Enforcing temporal constraints for LLM agents. *arXiv:*2512.23738; 2025.[DOI: 10.48550/arXiv.2512.23738]

[7] Zhang Y, Emma SY, En ALJ, Dong JS. RvLLM: LLM runtime verification with domain knowledge. *arXiv:*2505.18585; 2025.[DOI: 10.48550/arXiv.2505.18585]

[8] Liu B, Jiang Y, Zhang X, et al. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv:*2304.11477; 2023.[DOI: 10.48550/arXiv.2304.11477]

[9] Kambhampati S, Valmeekam K, Guan L, et al. LLMs can't plan, but can help planning in LLM-modulo frameworks. *arXiv:*2402.01817; 2024.[DOI: 10.48550/arXiv.2402.01817]

[10] Ji Z, Lee N, Frieske R, et al. Survey of hallucination in natural language generation. *ACM Comput Surv.* 2023;55(12):1-38.[DOI: 10.48550/arXiv.2202.03629]

[11] Pnueli A. The temporal logic of programs. *In: Proc. FOCS;* 1977. p. 46-57.[DOI: 10.1109/SFCS.1977.3]

[12] De Giacomo G, Vardi MY. Linear temporal and dynamic logic on finite traces. *In: Proc. IJCAI*; 2013. p. 854-60.

[13] Fuggitti F. LTLf2DFA [software]. Version 1.0.3. *Zenodo;* 2019. doi:10.5281/zenodo.3888410. Available from: https://github.com/whitemech/LTLf2DFA

[14] Rothkopf R, Zeng HT, Santolucito M. Enforcing temporal constraints on generative agent behavior with reactive synthesis. *arXiv:*2402.16905; 2024.[DOI: 10.48550/arXiv.2402.16905]

[15] Karpathy A. Notes on language models as structured knowledge bases. *GitHub gist;* Apr 2026. Available from: https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f

[16] Du Z, Qian Y, Liu X, et al. GLM: General language model pretraining with autoregressive blank infilling. *In: Proc. ACL;* 2022. p. 320-35.[DOI: 10.18653/v1/2022.acl-long.26]

[17] Wang L, Ma C, Feng X, et al. A survey on large language model based autonomous agents. *Front Comput Sci.* 2024;18(6):186345.[DOI: 10.48550/arXiv.2308.11432]

[18] Rajpurkar P, Chen E, Banerjee O, Topol EJ. AI in health and medicine. *Nat Med.*

2022;28:31-8.[DOI: 10.1038/s41591-021-01614-0]