Evaluating ReAct-Based LLM Reasoning for Korean Date and Schedule Processing

Yein Oh
Department of Computational Medicine
Ewha Womans University
Seoul 03760, Republic of Korea
yioh@ewha.ac.kr

Jihyun Jung, Sooyeon Park, Jisu Kim
Department of Artificial Intelligence and Software
Ewha Womans University
Seoul 03760, Republic of Korea
jh.jung@ewha.ac.kr, soopark723@gmail.com, rlawltn0214@ewha.ac.kr

Abstract

This study investigates whether a Large Language Model (LLM) employing the ReAct pattern—an iterative **Reason + Act** reasoning framework—outperforms a strong prompt-engineered baseline in Korean date and schedule processing tasks. Korean temporal expressions often combine relative time references (e.g., "the Friday after next") with complex scheduling constraints such as weekday restrictions, holiday exclusions, and minimum intervals. We evaluate three temporal reasoning subtasks—(T1) date normalization with phrases, (T2) date normalization with sentences, and (T3) constraint-based schedule generation—and compare ReAct-style reasoning loops against conventional few-shot Chain-of-Thought prompting across three dimensions: accuracy, latency, and token cost. The source code and datasets are available on GitHub[1].

1 Introduction

Natural language-based date computation and temporal normalization are essential capabilities for chatbots, calendar assistants, and workflow automation systems. In Korean, users frequently express time using relative temporal expressions (e.g., "the Friday after next," "three days after this weekend") and compound scheduling constraints such as weekday restrictions, holiday exclusions, or minimum intervals. Accurately interpreting and converting such expressions into absolute dates (e.g., YYYY-MM-DD) or generating lists of dates that satisfy all constraints remains a challenging task(Kim et al., 2025; Jeong et al., 2022). Recent large language models (LLMs) have shown promising results in temporal reasoning using prompt-engineering strategies such as few-shot examples, Chain-of-Thought (CoT) prompting, and structured JSON output formats. However, these prompt-based approaches often struggle to enforce multiple logical or temporal constraints consistently, particularly in complex scheduling scenarios. Such inconsistencies limit their reliability in practical, production-level systems(Cherukuri, 2025). The ReAct pattern (Reason + Act) offers a potential remedy by structuring the reasoning process into an explicit loop of thought → action → observation → reflection. Instead of generating a single monolithic reasoning chain, a ReAct-based model performs intermediate reasoning steps, applies rule-like actions (e.g., date calculation or constraint validation), observes the results, and updates its reasoning accordingly. This process allows for more systematic and verifiable temporal reasoning, without requiring explicit external code execution(IBM; Agent Patterns). This study investigates whether a ReAct-style reasoning loop can enhance large language model (LLM) performance on Korean temporal reasoning tasks compared to a CoT prompt-based baseline. We evaluate three subtasks—(T1) Date Normalization, (T2) Date Normalizaiton With Sentences, and (T3) Constraint-based Schedule Generation —and compare models along accuracy, constraint satisfaction, and computational efficiency (latency and token cost).

2 Related Works

2.1 Korean Temporal Normalizaton and Scheduling

With the development and proliferation of LLMs, recognizing temporal expressions in user queries is becoming increasingly important. This is because when a user asks, "At what age did Obama win the Nobel Peace Prize?" the LLM must be able to understand what 'Obama's age' and 'the year he won the Nobel Peace Prize' signify and calculate the interval between the two events(Piryani et al., 2025). Furthermore, in an era of rising expectations for AI assistants due to advancements in AI Agents, complex temporal expression recognition is also nec-

[1] https://github.com/yeinoh36/cot_or_react

essary. To resolve contextual expressions like "the last time we discussed this topic," LLMs require event memory and temporal expression recognition. However, LLM development has historically been heavily skewed towards English, and this focus has limited its applicability to other languages with unique linguistic characteristics, such as Korean(Park et al., 2024). Korean has a unique syntactic and semantic system different from English, presenting different challenges for LLM development. To address these issues, research is needed to determine whether ReAct-based reasoning can improve performance in Korean temporal reasoning.

2.2 ReAct and Structured Reasoning

The ReAct framework integrates explicit reasoning with environmental interaction through an iterative Thought → Action → Observation loop(Yao et al., 2022). Unlike traditional Chain-of-Thought (CoT) prompting, which performs all reasoning internally before producing an answer, ReAct interleaves reasoning steps with external or pseudo-actions (e.g., retrieving, checking, or verifying information). This design allows the model to refine its reasoning based on intermediate feedback, leading to more grounded and interpretable outcomes. ReAct was originally proposed to overcome key limitations in earlier prompting paradigms—namely, hallucination in CoT reasoning and rigid, memoryless decision-making in purely action-based models. By expanding the model's action space to include both linguistic reasoning actions ("thoughts") and environmental actions, ReAct enables flexible goal decomposition, progress tracking, and self-correction within a single framework. These findings suggest that structured reasoning–acting cycles can help language models handle tasks requiring both logical consistency and situational awareness. Building on this insight, the present study explores whether a ReAct-style reasoning loop can enhance Korean temporal reasoning – a domain that involves linguistically rich, constraint-heavy scheduling problems where intermediate reasoning and validation steps may play a crucial role.

3 Tasks

Korean temporal reasoning requires interpreting relative expressions, grounding them in context, and generating schedules that satisfy multiple interacting constraints. To systematically evaluate how the CoT baseline and the ReAct agent differ in robustness, we define three tasks that increase in linguistic and computational complexity. Each task isolates a distinct dimension of temporal reasoning—arithmetic, contextual grounding, and constraint satisfaction—allowing clear attribution of model strengths and failure modes.

3.1 Task 1: Date Normalization (T1)

T1 requires transforming a standalone relative temporal expression into an absolute calendar date (YYYY-MM-DD). These expressions range from simple weekday shifts or month-boundary references to cases requiring multiple sequential operations. For example, expressions such as "the first weekday after the end of this month" require identifying the month-end boundary before selecting the next available weekday. Likewise, expressions such as "the first weekday after the weekend immediately preceding the end of this month" involve locating a reference weekend and then computing the following weekday.

3.2 Task 2 : Date Normalization With Sentences (T2)

T2 extends the normalization problem by embedding temporal expressions within full sentences rather than presenting them as isolated phrases. Unlike T1, where the model receives a single self-contained expression, T2 requires identifying the correct temporal reference within a larger linguistic structure that may include multiple clauses, events, or distractor phrases. For example, whereas T1 provides an isolated phrase such as "next Friday," T2 may present the same expression within a full sentence such as "Could you tell me the date for next Friday?" requiring the model to first locate the embedded temporal phrase before normalization. The model must therefore identify the relevant span, disambiguate its scope, and apply the corresponding sequence of date transformations. The required output remains a single normalized date in YYYY-MM-DD format.

3.3 Task 3: Constraint-based Schedule Generation (T3)

T3 generalizes the temporal reasoning problem by requiring the model to generate a sequence of dates that jointly satisfy multiple explicit scheduling constraints. Unlike T1 and T2, which return a single normalized date, T3 produces an ordered list of dates whose validity depends on simultaneously

meeting conditions such as weekday restrictions, interval requirements, count constraints, holiday exclusions, or month-boundary limitations. Inputs frequently specify combinations such as producing three dates "every two weeks," generating occurrences "only on weekdays," or excluding national holidays during the generation process. For example, an instruction may request a sequence beginning on a specified anchor date and continuing at fixed intervals, or may restrict valid outputs to particular weekdays while requiring a minimum number of occurrences. In such cases, the model must iteratively evaluate candidate dates, filter out those that violate constraints, and continue generating until all conditions are satisfied. This task therefore requires long-horizon temporal reasoning, iterative validation, and the integration of multiple interacting rules rather than a single arithmetic transformation. The output of T3 is a list of normalized dates in YYYY-MM-DD format.

4 Methodology

4.1 Baseline Model: CoT Prompting

The baseline system adopts a standard few-shot Chain-of-Thought (CoT) prompting strategy. For each task, the model receives a fixed prompt containing (i) a small number of demonstrations illustrating the expected reasoning format and (ii) an instruction to derive the final answer after producing an explicit reasoning trace. All computations are performed internally by the language model: the CoT method does not rely on external tools, symbolic operations, or post-hoc validation. We use a consistent CoT prompting style across all tasks, although each task includes instructions tailored to its specific input and output format (a single date for T1 and T2, or a sequence of dates for T3). In all cases, the prompts share the same general structure: demonstration examples, a directive to reason step-by-step, and a final answer field.

4.2 ReAct Model: Thought → Action → Observation Loop

The ReAct agent is implemented as a multi-stage reasoning system that alternates between internal deliberation ("thought"), explicit tool invocation ("action"), and tool-returned feedback ("observation"). For T1, the agent performs a single-step decision: the "thought" module interprets the temporal expression and selects a tool such as calculator or calendar_db. The result returned in the observation is immediately converted into a final date by a lightweight decision module. For T2, the agent again selects a single tool call, but must first resolve the temporal span embedded in the input sentence. The thought module identifies which part of the sentence carries temporal meaning, reformulates it into a tool-compatible canonical query, and issues a corresponding action. The observation then directly produces the final normalized date, as T2 does not require iterative decision-making. For T3, the agent engages in a genuinely iterative process. The thought module acts as a planner, selecting atomic actions such as identifying start dates, applying intervals, or checking violations. Each action yields an observation, which a decision component evaluates against constraints (e.g., weekday rules or holiday exclusions). The resulting structured state is fed back into the planner, forming a closed control loop that repeats until all constraints are satisfied and a final schedule is generated. Across all tasks, tool interactions follow the same set of primitives defined in the implementation:

> Calculator for date arithmetic (day/week/month offsets, next/last weekday, interval advancement),
>
> Calendar_db for retrieving official holidays, solar terms, or anniversaries, and
>
> Search for external events without deterministic rules.

The modular separation between planning, tool execution, and observation evaluation enables the agent to maintain a structured record of partial results and constraints across steps and supports multi-step reasoning in T3, while keeping T1 and T2 efficient through single-step decision patterns.

5 Experiments

5.1 Datasets

We evaluate our models on three task-specific datasets constructed to reflect increasing levels of temporal reasoning difficulty: T1, T2, and T3. All datasets are annotated with an anchor date that defines the temporal reference point for interpretation. T1 consists of short temporal expressions such as "next Friday," "3 days later," or "the weekday before the last day of this month." Each instance contains a single relative expression, and the output is a single normalized ISO-8601 date.

This dataset tests whether a model can map compact temporal cues to deterministic date transformations. T2 extends the complexity by embedding similar temporal expressions inside full sentences, such as "Please tell me the date for next Friday," or "Can you tell me what date it will be in three days?" The annotation identifies the sentence-level temporal span and its corresponding normalized date. T3 is a multi-step scheduling dataset in which the user specifies constraints such as interval rules, minimum count requirements, weekday or holiday exclusions, and external event references. Each instance contains: (1) an anchor date, (2) the textual instructions, (3) structured constraints (e.g., interval_days, specific_weekdays, exclude_holidays), and (4) the gold standard consisting of one or more dates satisfying all constraints. Many cases require identifying external events (e.g., concerts, exams) via the search tool, applying holiday filters via the calendar_db tool, and iteratively generating candidate dates using calculator-based arithmetic. The dataset therefore evaluates multi-step planning, tool selection, and constraint satisfaction. Across all datasets, annotations were manually inspected to ensure deterministic gold standards, and ambiguous expressions were removed. The datasets collectively cover a broad range of temporal constructs, from simple relative expressions to complex multi-constraint scheduling instructions.

5.2 Experimental Settings

All experiments were conducted under a controlled evaluation pipeline to ensure comparability between the CoT baseline and the ReAct agent. Both methods were evaluated on the same inputs, with identical anchor dates, task definitions, and output formats. Two LLM backbones were used for all executions: the *GPT-4.1-mini* model accessed through the *OpenAI* API and the *SOLAR-pro2* model accessed through the *Upstage* API. No decoding parameters were manually overridden; consequently, all generations followed each model's default API settings. This setup ensures that any observed performance differences arise solely from the contrasting reasoning strategies—CoT versus ReAct—rather than from differences in sampling configuration or decoding behavior. The CoT baseline used a fixed prompt containing task-specific instructions and few-shot demonstrations, and the model produced a single forward pass without tool interaction. The ReAct agent followed the Thought $\rightarrow$ Action $\rightarrow$ Observation loop described in Section 4.2, invoking external tools when required. T1 and T2 required only a single reasoning cycle, whereas T3 involved multiple cycles depending on the number of constraints and intermediate validation steps. For T3, the ReAct implementation enforced a maximum of 10 reasoning turns to prevent unbounded iteration when constraints could not be satisfied.

5.3 Evaluation Metrics

We evaluate both CoT and ReAct using accuracy metrics and computational metrics that reflect the requirements of temporal reasoning tasks. For T1 and T2, each model prediction is a single normalized ISO-8601 date. Performance is measured using exact-match accuracy, where a prediction is counted as correct only if it matches the gold standard date string exactly. Partial matches or off-by-one errors are not credited, as the tasks require fully resolved temporal normalization. For T3, the output is a sequence of dates that must satisfy all constraints. A prediction is considered correct only if the entire generated list matches the gold schedule exactly in both content and order. Any deviation—including missing or extra dates, incorrect intervals, or constraint violations—results in failure. If the ReAct agent exceeds the maximum reasoning depth (10 turns) without producing a valid schedule, the instance is also marked as a failure. In addition to correctness, we report latency and token usage for each task. Latency is measured as the wall-clock time from the initial prompt to the final output, capturing the computational overhead of multi-step reasoning in T3 relative to the single-step patterns in T1 and T2. Token usage includes both prompt and completion tokens, allowing comparison of the computational cost of CoT and ReAct.

6 Results and Analysis

6.1 T1 Performance

In the T1 task, clear differences emerged across model configurations in both performance and computational costTable 1. GPT-CoT achieved 72.20% accuracy, whereas GPT-ReAct improved this to 76.20%.A McNemar test confirmed that this gain was statistically significant ($p = .0018$), indicating that ReAct prompting yields meaningful error reduction within GPT models. In contrast, Solar-CoT produced the lowest accuracy (66.00%), but it exhibited the best computational efficiency, requiring only 1.68 seconds per query and generating approx-

imately 1376 tokens. Solar-ReAct reached 76.20% accuracy, matching the absolute performance level of GPT-ReAct; however, this improvement was not statistically significant relative to its own CoT baseline ($p > .05$). Notably, Solar-ReAct maintained a modest computational footprint—2.6–2.8 seconds latency and 2.3–2.4k tokens per query—making it the most balanced configuration in terms of accuracy gain without substantial cost escalation. Meanwhile, although GPT-ReAct achieved the highest accuracy, it required 3.4 seconds per sample and generated approximately 25,000 tokens, resulting in the lowest cost-normalized efficiency among T1 models.

Table 1: Performance Comparison on T1 Task

Model	Accuracy	Avg Latency(s)	Avg Token Usage
GPT-CoT	72.20%	∼5.11	∼2,388
GPT-ReAct	76.20%	∼3.40	∼25,000
Solar-CoT	66.00%	∼1.68	∼1,376
Solar-ReAct	76.20%	∼2.6–2.8	∼2,350–2,450

6.2 T2 Performance

In T2, clear variations were observed across model configurations in both accuracy and computational costTable 2. GPT-CoT achieved 74.40% accuracy, while GPT-ReAct improved performance to 79.00%. A McNemar test confirmed this improvement to be statistically significant ($p = .030$), suggesting that ReAct prompting provides meaningful error reduction for sentence-level temporal reasoning within GPT models. Solar-CoT obtained the lowest accuracy (66.70%) but exhibited the highest computational efficiency, requiring only 1.61 seconds of latency and approximately 1376 tokens. Solar-ReAct improved accuracy to 74.85%, reaching a performance level comparable to GPT-CoT; however, paired testing did not show a statistically significant difference relative to Solar-CoT ($p > .05$). Nonetheless, Solar-ReAct maintained modest computational cost—2.5–3.0 seconds latency and 2.3–2.5k tokens per sample—making it the most balanced configuration with respect to accuracy gain and efficiency preservation. In contrast, GPT-ReAct achieved the highest accuracy (79.00%) yet incurred substantial computational overhead, including 3.25 seconds latency and 25,000 tokens generated per query, again producing the lowest cost-normalized efficiency among T2 models.

Table 2: Performance Comparison on T2 Task: Accuracy, Latency, and Token Usage

Model	Accuracy	Avg Latency(s)	Avg Token Usage
GPT-CoT	74.40%	∼5.32	∼2,393
GPT-ReAct	79.00%	∼3.25	∼25,000
Solar-CoT	66.70%	∼1.61	∼1,376
Solar-ReAct	74.85%	∼2.5–3.0	∼2,350–2,460

Table 3: Performance Comparison on T3 Task: Accuracy, Latency, and Token Usage

Model	Accuracy	Avg Latency (s)	Avg Token Usage
GPT-CoT	22.20%	∼1.55	∼768
GPT-ReAct	21.40%	∼38–50	∼18,000
Solar-CoT	16.80%	∼2.85	∼1,350
Solar-ReAct	21.40%	∼25–30	∼21,000–30,000

6.3 T3 Performance

T3 was the most challenging task, with all models producing their lowest performance hereTable 3. GPT-CoT achieved 22.20% accuracy, and GPT-ReAct produced 21.40%, showing virtually no difference. A McNemar test corroborated the absence of statistical significance ($p \approx 1.00$), indicating that prompting architecture does not improve accuracy for multi-event temporal reasoning within GPT. Solar-CoT recorded 16.80% accuracy—the lowest among all models—but demonstrated relatively efficient computation, requiring 2.85 seconds latency and approximately 1,350 tokens. Solar-ReAct improved accuracy to 21.40%, but the paired comparison did not reveal statistical significance ($p > .05$). Moreover, this gain came at a sizable computational cost: Solar-ReAct averaged 25–30 seconds latency and 21,000–30,000 tokens, resulting in limited efficiency relative to its accuracy improvement. GPT-ReAct imposed the largest computational penalty. It required 38–50 seconds per query and generated more than 18,000 tokens, as repeated tool failures and self-repair cycles greatly extended the reasoning trajectory. However, the resulting accuracy was 21.40%, and no statistically significant improvement was observed, making GPT-ReAct the worst trade-off configuration—high cost without measurable performance gain.

6.4 Error Analysis

Error inspection revealed distinct failure modes between reasoning strategies. CoT exhibited recurrent ±1-day drift on relative date shifts, inconsistencies in Korean week boundary interpretation, mismatches between reasoning traces and final outputs,

and premature reasoning termination—suggesting instability in calendar arithmetic grounding. By contrast, ReAct failures were dominated by tool-driven instability, including turn-limit collapses, redundant re-evaluation of events, and incomplete constraint chaining. These issues were most pronounced in T3, where tool-guided reasoning amplified error density rather than alleviating it, implying that tool invocation alone does not guarantee structured planning under multi-event reasoning conditions.

6.5 Ablation Study

To identify whether the failure of the original Solar-ReAct system was attributable to prompting limitations or to its rigid tool specification, we conducted an ablation experiment in which the hand-coded tool layer was replaced with an LLM-driven reasoning module. The original Solar-ReAct model achieved 21.40% accuracy on T3, despite low latency (2.5–3.0 s) and limited token consumption (2.4k tokens per query). Inspection of wrong cases revealed that most failures originated from non-executable tool calls, misaligned argument formats, or premature halting when tools returned no state, suggesting that the architecture failed not because of its reasoning trajectory, but because the agent could not meaningfully use its toolkit. Replacing hand-coded tools with an LLM-based tool dramatically altered performance characteristics. The modified Solar-ReAct achieved 31.40% accuracy, a relative improvement of +46.7% over its original configuration. However, this gain came with substantial computational cost: average latency increased to 25.14 seconds per instance, and token consumption rose to approximately 24K tokens per query—nearly matching the computational footprint of GPT-ReAct.

7 Conclusion

7.1 Summary of Findings

Our evaluation across three temporal reasoning tasks (T1, T2, T3) using varying prompting strategies (CoT vs. ReAct) and model sizes (GPT vs. Solar) yields four primary findings: (i) ReAct Efficacy and Task ComplexityReAct prompting significantly outperforms CoT in simpler, single-event or sentence-level tasks (T1, T2) for GPT models, showing statistically significant error reduction ($p < .05$). However, this advantage disappears in complex document-level reasoning (T3). In high-complexity scenarios, neither prompting strategy proved effective, with ReAct merely increasing computational overhead without improving accuracy. (ii) The Cost-Performance "Sweet Spot" Solar-ReAct emerged as the most balanced configuration for T1 and T2. It matched the accuracy of GPT-ReAct in single-event grounding (76.20%) while maintaining a computational footprint comparable to standard CoT models (low latency and 2.4k tokens). This suggests that for tasks with moderate complexity, mid-sized models with structured prompting can replace larger models at a fraction of the cost.3. The Agentic Trade-off in Complex TasksFor the most challenging task (T3), the ReAct framework induced a negative performance-cost loop. Both GPT and Solar models suffered from "tool-driven instability," where the agentic approach led to excessive token consumption (up to 30k–50k tokens) and high latency (up to 50s) due to repetitive self-repair cycles and tool failures, without yielding statistically significant accuracy gains over the much cheaper CoT baselines.

7.2 Implications

These results emphasize that evaluation of reasoning frameworks should consider not only accuracy but also cost-performance dynamics and task-structure alignment. ReAct appears suitable for lightweight single-event reasoning where cost-efficiency matters; however, multi-event temporal reasoning requires explicit event segmentation, constraint propagation, and hierarchical planning. The ablation findings highlight that tool-enhanced prompting is not inherently beneficial—reasoning stability depends on orchestration rather than capability injection.

7.3 Limitations & Future Works

This work primarily relied on accuracy-based evaluation; qualitative analysis of reasoning traces and failure propagation is left for future research. Moreover, T1/T2 tasks contain simpler temporal structures, and task complexity differences may have contributed to performance discrepancies. Future directions include developing multi-event-oriented pipeline architectures, exploring adaptive tool-use policies and recovery mechanisms, designing evaluation metrics for temporal reasoning stability and extending analysis to richer expressions of Korean temporal language.

8 Contributions of team members

Yein Oh: Solar Pro 2 ReAct 실험, 발표

Jisu Kim: GPT CoT 실험, Latex 편집

Jihyun Jung: GPT ReAct 실험

Sooyeon Park: Solar Pro 2 CoT 실험, PPT 제작

References

Agent Patterns. React - agent patterns. https://agent-patterns.readthedocs.io/en/stable/patterns/re_act.html. Accessed: 2025-10-21.

Milind Cherukuri. 2025. Cost, complexity, and efficacy of prompt engineering techniques for large language models. *IJSAT-International Journal on Science and Technology*, 16(2).

IBM. The react agent: A new framework for ai. https://www.ibm.com/think/topics/react-agent. Accessed: 2025-10-21.

Young-Seob Jeong, Chaegyun Lim, SeungDong Lee, Medard Edmund Mswahili, Goodwill Erasmo Ndomba, and Ho-Jin Choi. 2022. Rule-based normalization of relative temporal information. *Journal of The Korea Society of Computer and Information*, 27(12):41–49.

Jinpyo Kim, Gyeongje Cho, Chanwoo Park, Jongwon Park, Jongmin Kim, Yeonkyoun So, and Jaejin Lee. 2025. Thunder-llm: Efficiently adapting llms to korean with minimal resources. *arXiv preprint arXiv:2506.21595*.

Chanjun Park, Hyeonwoo Kim, Dahyun Kim, Seonghwan Cho, Sanghoon Kim, Sukyung Lee, Yungi Kim, and Hwalsuk Lee. 2024. Open ko-llm leaderboard: Evaluating large language models in korean with ko-h5 benchmark. *arXiv preprint arXiv:2405.20574*.

Bhawna Piryani, Abdelrahman Abdullah, Jamshid Mozafari, Avishek Anand, and Adam Jatowt. 2025. It's high time: A survey of temporal information retrieval and question answering. *arXiv preprint arXiv:2505.20243*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.