*Journal of Methodical Computing and Communications* **(JMCC)**, Vol. 2026, pp. 1-20, 2026.

# Structural Prompt Engineering in Constitutional AI Models: A Technical Analysis of Claude's Techniques

Ali A. Alsaffar[1*]

[1] Department of Business Administration, College of Administration and Economics, University of Mosul, Mosul, Iraq

***Corresponding email:** ali_alsaffar@uomosul.edu.iq

**Received** 13 September 2026**; Revised** 15 October 2026**; Accepted** 30 October 2026**; Published** 5 November 2026

**Abstract:** This paper examines structural prompt engineering for Claude, a large language model trained by Anthropic with Constitutional AI (CAI). We formalize a four-component procedure, Constitutional AI-Aligned Structured Prompting (CASP), comprising explicit role and context framing, XML-based input structuring, explicit step-by-step reasoning, and output control through schema-constrained structured outputs, with prompt caching for reused contexts. CASP is specified as an algorithm and evaluated in a controlled ablation study on two public benchmarks: entity extraction to JSON on CoNLL-2003 (T1; 50 items, also run with an embedded conflicting instruction) and multi-step arithmetic on GSM8K (T2; 50 items), using Claude Sonnet 4.6. The full procedure, a naive baseline and leave-one-out ablations were compared using automatic gold-based scoring and Holm-corrected paired Wilcoxon tests. Full CASP achieved a typed entity F1 of 0.855 on T1 versus 0.000 for a naive baseline that did not specify an output format ($p < 0.001$), because none of the baseline's prose answers was machine-parseable; when these prose answers were parsed post hoc, CASP remained 0.097 higher ($p = 0.005$). On T2, all conditions scored 0.960 except when explicit reasoning was removed (accuracy 0.740, $p = 0.017$, $r = 0.846$), because a response schema without a reasoning field suppressed the model's spontaneous reasoning. XML showed no significant advantage over Markdown. No condition actually obeyed the embedded instruction. Prompt caching reduced input cost by 85% with no significant change in output quality ($p = 1.000$, output agreement 0.93). Code, sampled data and raw outputs are released for replication.

***Keywords:*** Prompt Engineering; Claude; Constitutional AI; XML-based Prompt Structuring; Chain-of-Thought Reasoning; Structured Outputs; Ablation Study.

## 1. Introduction

Large Language Models (LLMs) have substantially advanced natural language processing, and their scale allows them to perform new tasks from only a few examples [1]. As a result, the way users formulate their requests has become an important factor in the quality of the results. This shift led to the emergence of prompt engineering as a new discipline, drawing its inspiration from computer science, computational linguistics, and user interface design [2].

Anthropic's Claude is trained with a methodology called Constitutional AI (CAI), in which the model critiques and revises its own outputs against a set of written principles [3] rather than relying solely on reward signals from human evaluation. This builds on reinforcement learning from human feedback (RLHF) [4, 5] but replaces part of the human feedback with AI feedback, an approach known as reinforcement learning from AI feedback (RLAIF). The Claude 3 technical report documented a 200,000-token context window [6]; as of September 2026, Anthropic documents context windows of up to one million tokens for its current Claude Opus and Sonnet models [7]. Because product specifications of this kind change quickly, all such figures in this paper are reported as of the date of the cited documentation.

Prompt engineering methods developed for other model families may not transfer directly to Claude, since LLM performance is sensitive even to formatting choices that carry no semantic content [8, 9]. Anthropic's documentation recommends specific conventions, most notably delimiting prompt components with XML tags [10]. This suggests, but does not establish, that XML-delimited structure is well represented in Claude's training or fine-tuning data. In this paper this statement is treated as a documentation-grounded hypothesis, H1 (Section 3.2), whose observable consequence is tested rather than assumed; other training-related statements are treated in the same way. Without a systematic account of these conventions, practitioners' risk generic answers, output that downstream programs cannot parse, and incomplete use of the available context.

To fill this gap, the current study has the following research objectives:

- To review the documented training approach and vendor guidance that motivate structured prompting for Claude.
- To formalize Claude-specific techniques (XML structuring, explicit reasoning, output control and prompt caching) as a replicable procedure, specified as an algorithm.
- To evaluate the procedure experimentally through a controlled ablation study on public benchmarks, reporting quantitative metrics and statistical tests.
- To compare Claude's documented prompting conventions with those of competing model families, keeping this documentation-based comparison separate from the measured results.

The main contribution of this work is a Claude-specific structured prompting procedure (CASP) together with a controlled evaluation of it. The procedure is a synthesis rather than a new technique: its individual components are established in the literature, and Table 2 (Section 2) maps each component to its closest prior foundation and states what the combination adds. CASP differs from existing prompt-engineering approaches in three respects: its components are combined into a single, ordered procedure (Algorithm 1, Section 3.6) rather than offered as independent recommendations; each step is tied to one model family's documented conventions rather than to model-agnostic advice; and the contribution of each component is measured separately through a leave-one-out ablation. The practical value of the combination is that the components address different failure modes. On the entity-extraction task, the full procedure raised typed F1 from 0.000 for a naive prompt that did not specify an output format to 0.855, because only the structured prompts produced machine-parseable output, and it remained about 0.1 higher when the naive prompt's prose answers were parsed post hoc; on the arithmetic task, explicit reasoning was the only component whose removal significantly lowered accuracy (from 0.960 to 0.740), because a response schema without a reasoning field suppressed the model's spontaneous reasoning; and prompt caching reduced input cost by about 85% without a measurable change in quality (Section 5). The evaluation addresses four research questions. RQ1: does full CASP improve accuracy and format compliance over a naive baseline? RQ2: what is the individual contribution of each component when it is removed from the full procedure? RQ3: does XML delimiting outperform Markdown delimiting of identical content, including when the input contains an embedded conflicting instruction? RQ4: does prompt caching affect output quality, as opposed to latency and cost?

## 2. Related Work

Work in foundational alignment has shown that fine-tuning on human preference signals improves instruction following beyond scaling alone [4, 5]. Anthropic's Constitutional AI approach (Bai et al. [3]) extends this, substituting a part of the human feedback loop with AI-generated feedback, which is assessed against a constitution document. Anthropic's guidance recommends explaining the context or motivation behind an instruction [10]. The author conjectures that this advice is related to constitutional training, because the model's refusal and caution behavior is grounded in written principles rather than in surface phrasing; this conjecture is not tested here. The role-and-context component of CASP (Section 3.1) operationalizes the recommendation, and its contribution is estimated in the ablation study of Section 5.2.

Other studies examine individual prompting techniques. Wei et al. [11] demonstrate the utility of directing a model through intermediate steps of inference — chain-of-thought prompting — to substantially boost performance on multi-step tasks. Section 3.3 describes how CASP applies this idea to Claude.

Reynolds and McDonell [12] find that a well-formulated instruction, even without examples, can outperform a traditional few-shot prompt, consistent with Anthropic's advice to be clear and direct [10]. White et al. [13] catalog reusable prompt patterns and treat prompt design as a software-engineering task, but their catalog is general-purpose and does not address Anthropic's documented recommendation of XML tags for Claude [10]. Zhou et al. [14], by contrast, automate prompt generation rather than designing prompts manually — an approach that complements, but differs from, our manually constructed structural method.

Beyond text instructions, Schick et al. [15] show that language models can learn to invoke external tools automatically. Tool use is outside the scope of CASP and is listed as future work in Section 5.7. On long contexts, Liu et al. [16] report an important warning: extended context windows show a U-shaped retrieval-accuracy curve, performing best when relevant information sits near the beginning or end of the input and worse when it sits in the middle. This has direct implications for how core instructions should be placed, and CASP builds the corresponding placement rule into its procedure explicitly (Algorithm 1, line 12).

**Table 1:** Summary of methods, datasets, models, and reported performance in the related work reviewed in Section 2.

| Author | Method | Dataset | Model | Performance and Evaluation |
|---|---|---|---|---|
| Bai et al. [3] (2022) | Constitutional AI (RLAIF): AI self-critiques and revises its own outputs against a written set of principles, replacing part of the human feedback loop | Model-generated helpfulness and harmlessness dialogues (self-generated, not a public benchmark) | Anthropic's own language model (precursor to Claude) | Human preference ratings showed harmlessness comparable to or better than RLHF baselines, with fewer evasive refusals |
| Christiano et al. [4] (2017) | Reward modeling learned from pairwise human comparisons of trajectory segments, used to train a policy via RL | Atari games and simulated robotics control tasks | Policy networks trained with standard RL algorithms (e.g., A2C, TRPO) | Matched the performance of RL trained on hand-engineered rewards while requiring far fewer human comparisons |
| Ouyang et al. [5] (2022) | Supervised fine-tuning on human demonstrations followed by RLHF (PPO) on human preference rankings | OpenAI API prompt distribution plus human-written demonstrations and comparisons | GPT-3 (1.3B, 6B and 175B parameter variants) | Outputs of the 1.3B InstructGPT model were preferred by human raters over the 175B GPT-3 model; improved truthfulness and reduced toxicity |
| Wei et al. [11] (2022) | Chain-of-thought prompting: few-shot exemplars that include intermediate reasoning steps before the final answer | Arithmetic (e.g., GSM8K), commonsense and symbolic reasoning benchmarks | PaLM (540B) and GPT-3 | Substantial accuracy gains on multi-step reasoning tasks, with PaLM 540B reaching state-of-the-art results on math word-problem benchmarks |
| Reynolds & McDonell [12] (2021) | Prompt programming: careful zero-shot instruction design without in-context examples | Illustrative qualitative examples (e.g., translation tasks); no formal benchmark suite | GPT-3 | Qualitative demonstration only: well-formulated zero-shot prompts matched or exceeded few-shot performance; no quantitative benchmark score reported |
| White et al. [13] (2023) | Prompt pattern catalog: reusable prompt patterns adapted from software design patterns | Not applicable (conceptual catalog, not evaluated on a dataset) | ChatGPT | Qualitative illustration of pattern usage; no quantitative performance evaluation |
| Zhou et al. [14] (2023) | Automatic Prompt Engineer (APE): LLM-generated candidate instructions, iteratively scored and selected | Instruction Induction task suite and selected BIG-Bench tasks | GPT-3 (text-davinci-002-class model) | Automatically generated prompts matched or exceeded human-written prompts on most evaluated tasks |
| Schick et al. [15] (2023) | Toolformer: self-supervised learning of when and how to call external APIs, inserted directly into generated text | Subset of CCNet for self-supervised training; evaluated on LAMA, math (ASDiv, SVAMP) and QA benchmarks | GPT-J (6.7B) | Outperformed the much larger GPT-3 (175B) in zero-shot settings on several downstream tasks |
| Liu et al. [16] (2024) | Empirical analysis of retrieval accuracy as a function of the position of relevant information within a long context | Multi-document question answering and synthetic key-value retrieval tasks | GPT-3.5-Turbo, Claude 1.3, MPT and LongChat | U-shaped accuracy curve: best performance when relevant information is near the start or end of the context, with degradation in the middle |
| Sahoo et al. [2] (2024) | Systematic literature survey and taxonomy of prompt engineering techniques | Not applicable (survey, not empirical) | Not applicable (covers multiple LLM families generically) | Not applicable; no empirical performance results reported |

Sahoo et al. [2] review prompt engineering across LLM families. Like most of this literature, their survey is not tied to a specific model, and it does not cover Claude's constitutional training or the vendor's XML convention. Taken together, the studies above supply the individual building blocks: alignment through constitutional or RLHF methods, chain-of-thought reasoning, prompt pattern catalogs and long-context limits. None of them,

however, combines these blocks into a procedure built around one model family's documented conventions. Section 3 addresses this gap.

A further issue is how the effectiveness of prompting is measured. Much of the prompting literature cited above, such as work on chain-of-thought prompting and automatic prompt generation, is evaluated on standard benchmarks that are scored automatically [11, 14], whereas alignment studies rely largely on human preference ratings [3, 5] and several prompting studies are qualitative [12, 13]. This suits algorithmic comparisons such as chain-of-thought reasoning or automated prompt search. It says less about the structured tasks for which XML tagging is usually recommended, such as extracting structured data. The present study therefore evaluates CASP on public benchmarks with reference-based automatic scoring (Sections 4 and 5), and keeps the documentation-based comparison with other vendors separate from the measured results.

Table 1 summarizes the works discussed above in terms of their method, dataset, model, and reported performance, making explicit the empirical basis (or lack thereof) of each contribution and setting up the comparison with CASP presented later in this paper. Two further strands of work, beyond those summarized in Table 1, inform the evaluation design. First, LLM performance is sensitive to formatting choices that carry no semantic content: Sclar et al. [8] report large accuracy differences across semantically equivalent prompt formats, and He et al. [9] find that switching among plain-text, Markdown, JSON and YAML templates changes performance on several benchmarks. This motivates comparing XML with Markdown delimiting of identical content, rather than only with an unstructured prompt. Second, when untrusted data are concatenated with instructions, models may follow instructions embedded in the data (indirect prompt injection) [20], and explicitly delimiting or marking the data is one proposed mitigation [21]. The embedded-instruction variant of the extraction task (Section 4.1) draws on this line of work. Table 2 makes this positioning explicit: it maps each of the four CASP components introduced in Section 3 onto its closest foundation in the literature reviewed above, states the scope or limitation of that prior work, and states what CASP adds on top of it. Rows 1–4 correspond one-to-one with Sections 3.1–3.4; the final row states the paper's overall contribution as the synthesis of these four dimensions into a single, replicable, Claude-specific method—a synthesis that, to the author's knowledge, no single prior work attempts.

**Table 2:** Positioning of the CASP framework's four components relative to their closest foundations in prior work.

| CASP component | Foundation in prior work | Scope / limitation of that prior work | What CASP adds |
|---|---|---|---|
| Explicit role and context framing (3.1) | Reynolds and McDonell's instruction-focused zero-shot design [12]; Anthropic's own prompting guidance [17] | Demonstrated qualitatively for zero-shot prompts in general; not tied to any model's specific training regime | Operationalizes clear, explicit instruction design as a concrete, replicable step inside a Claude-specific framework |
| XML-based input structuring (3.2) | White et al.'s prompt pattern catalog [13] | General-purpose and cross-model; does not address Anthropic's documented recommendation of XML tags for Claude [10] | Adopts the XML-tagging convention documented for Claude [10] as a fixed step, treats its link to training as hypothesis H1, and tests its benefit against Markdown on identical content (Section 5.3) |
| Step-by-step explicit reasoning (3.3) | Wei et al.'s chain-of-thought prompting [11] | Model-agnostic; not linked to any model's native reasoning interface | Implements chain-of-thought as an explicit reasoning block requested in the prompt (<thinking>/<answer> tags) or as a reasoning field placed before the answer in a structured-output schema |
| Output control via prefilling and caching (3.4) | Anthropic's own technical documentation [18] | A practical vendor technique with no prior treatment inside an academic prompt-engineering framework | Integrates output control (prefilling on models that support it; schema-constrained structured outputs on current models [19]) and prompt caching as explicit, replicable components of a structured method |
| Overall synthesis | Bai et al. [3] (constitutional training), Liu et al. [16] (long-context retrieval), Sahoo et al. [2] and Zhou et al. [14] (general surveys / automated prompting) | Each address one dimension (alignment, long-context limits, or cross-model cataloguing) in isolation | To the author's knowledge, CASP is the first procedure to combine these dimensions into one replicable, Claude-specific method, and it is evaluated experimentally here (Sections 3–5) |

## 3. Proposed Method

This section presents a four-component prompting procedure called Constitutional AI-Aligned Structured Prompting (CASP), designed for direct use by practitioners. Each component is motivated by a documented vendor recommendation or by a documentation-grounded hypothesis about Claude's training (Section 2). Sections 3.1–3.4 describe the components, Section 3.5 applies them to a second task, and Section 3.6 specifies the complete procedure as Algorithm 1.

### *3.1 Identifying the Role and Context Explicitly*

Anthropic's guidance recommends treating Claude like a competent new employee who does not yet know the task's specific context [10]. In practice, every instruction should establish three things clearly: the exact task, the intended audience, and any constraints stated in concrete, measurable terms rather than vague qualifiers — an exact word count, for instance, rather than a request for a "summary." Instructions should also state what the model should do rather than what it should not do, since negative framing is more ambiguous and more likely to produce unintended behavior [10]. A vague request such as "Summarize this" becomes: "Write an executive summary of 120–150 words for a company president" — task, audience, and constraint are now all explicit. Figure 1 contrasts a vague instruction with an explicit one along these three dimensions.

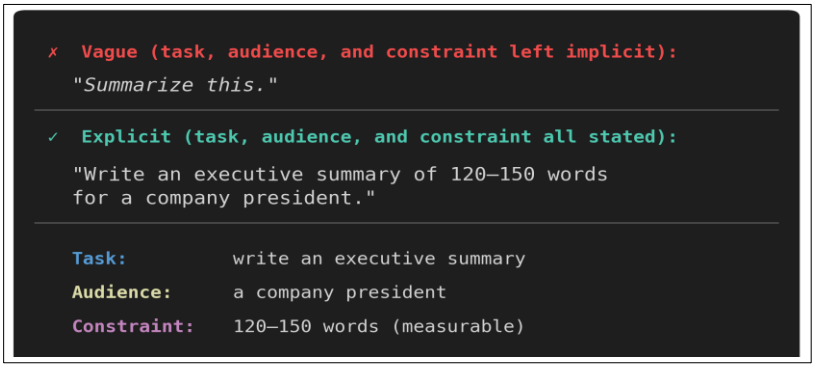



**Figure 1:** A vague instruction versus an explicit one, contrasted by task, audience, and constraint.

### *3.2 Structuring Inputs Explicitly with XML*

Anthropic's documentation recommends wrapping distinct prompt components in explicit tags such as <context>, <instructions> and <examples> [10]. We hypothesize (H1) that this recommendation reflects XML-delimited structure in Claude's training or fine-tuning data, and that XML delimiting therefore helps the model separate instructions from reference data. H1 cannot be observed directly from outside Anthropic; Section 5.3 tests its observable consequence by comparing XML with Markdown delimiting of identical content. Applying the component takes three steps: (a) identify each independent functional block in the instruction, such as the system role, reference context, instructions, or examples; (b) enclose each block in a pair of clearly named tags; and (c) refer to these tag names explicitly in the instruction text (e.g., "Based on the text within the <context> tags"). A simple example is shown in Figure 2.




**Figure 2:** A simplified XML-tagged prompt with role, context, and instruction blocks.

This separates the assistant's persona, the reference material and the task cleanly.

### *3.3 Step-by-Step Explicit Reasoning*

For multi-step logic or numerical reasoning tasks, CASP instructs Claude to reason first in a separate block and then give the final answer in a separate block. This is a model-specific implementation of the chain-of-thought findings of Wei et al. [11]. A typical instruction is: "Reason step by step inside <thinking> tags, then write only the final number inside <answer> tags." Separating the reasoning from the final output allows an application to strip the reasoning programmatically before returning the answer, which is useful in production settings where long reasoning traces should not be exposed to the end user. Current Claude models also offer native (adaptive) thinking; as of September 2026, Anthropic describes manual tag-based reasoning as a fallback for when native thinking is disabled [10]. In the experiments, native thinking was not requested, so that the contribution of this component could be isolated (Section 4.3). Figure 3 shows a worked example of this pattern.

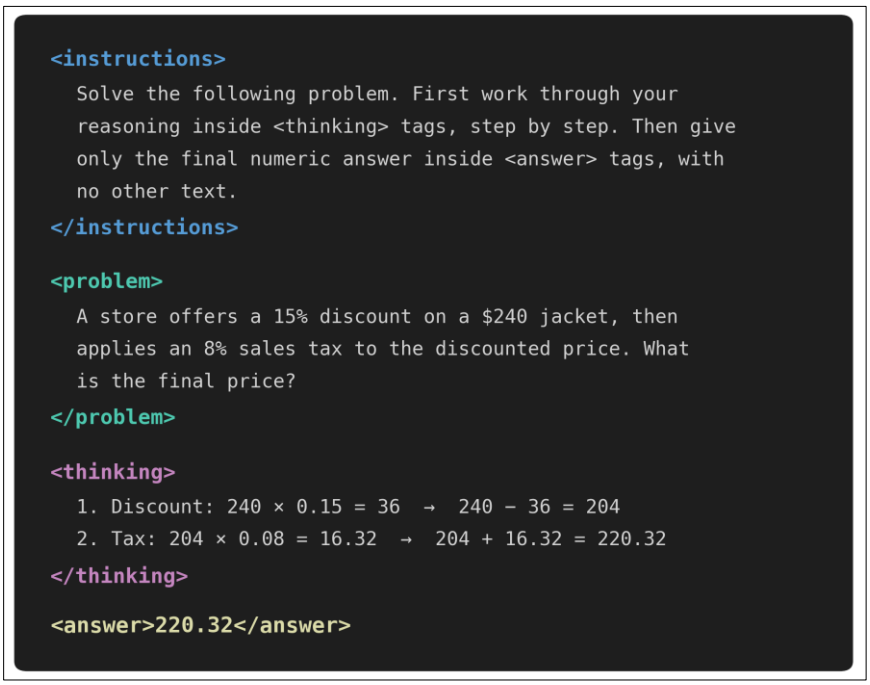



**Figure 3:** Separating step-by-step reasoning (<thinking>) from the final answer (<answer>) for programmatic stripping.

### *3.4 Output Control: Prefilling, Structured Outputs and Prompt Caching*

For output in a strict format such as JSON, CASP constrains the response from its first token so that no conversational framing precedes the structure. Two mechanisms implement this, depending on the model. On models that support it, the assistant turn is prefilled so that the response begins mid-structure, at an opening brace ("{"), instead of with an opener such as "Sure, here's the JSON:". As of September 2026, however, prefilling the final assistant turn is no longer supported from the Claude 4.6 generation onward, and such requests return an error [10]. On these models CASP instead uses schema-constrained structured outputs (the output_config. format parameter), which restrict decoding to a supplied JSON schema [19]. In the experiments of Section 5, structured outputs are evaluated on Claude Sonnet 4.6; prefilling was not tested, because the Claude Haiku 4.5 arm of the design was not run (Section 5.7).

For workflows that repeatedly reuse large, fixed contexts, such as a long reference document queried many times, CASP also recommends marking that content as cacheable. Anthropic documents substantial reductions in cost and response time for such workloads [18]; whether caching also changes output quality is tested directly in Section 5.4. Figure 4 shows a request that combines caching, marked with the cache_control parameter on the fixed context block, with schema-constrained structured outputs, and notes the prefilling alternative for earlier models.

```
# Request body (Anthropic Messages API)
"system": "You are a documentation assistant. Your output is
          parsed by a program, so return JSON only."
"messages": [{ "role": "user", "content": [
  { "type": "text",
    "text": "<context>[... long reference document
             queried many times, e.g., a 40-page product manual ...]
             </context>",
    "cache_control": {"type": "ephemeral"} },  # cached prefix
  { "type": "text",
    "text": "<instructions>Extract the product name,
             model number, warranty period and power rating from
             the manual inside the <context> tags.</instructions>" }
]}],
"output_config": {"format": {"type": "json_schema", "schema": {
   "type": "object", "required": ["product","model","warranty","power"],
   "properties": {... four string fields ...},
   "additionalProperties": false }}}   # structured outputs

Earlier models only (prefill not supported from Claude 4.6 on):
assistant turn prefilled by the developer with  {
```

**Figure 4:** Combining prompt caching (cache_control on the fixed context block) with output control in a single request: schema-constrained structured outputs on current models, and response prefilling on earlier models that still support it (Section 3.4).

### *3.5 Application to a Second Task and Replicability*

To show that CASP is not limited to the structured-data-extraction task evaluated in Section 5, the template below applies the same goes four components to a different task: helping a researcher review a scientific manuscript (Figure 5).

```
<system>
  You are a peer reviewer with expertise in computer science
  journals, providing constructive and specific comments for
  the authors of a submitted manuscript.
</system>

<context>
  [Summary text of the manuscript and its Methodology section]
</context>

<instructions>
  1. Inside <thinking> tags, identify up to three methodological
     weaknesses in the manuscript text within <context>.
  2. For each weakness, state (a) what is missing or unclear and
     (b) a concrete way the authors could address it.
  3. Inside <answer> tags, present the final comments as a
     numbered list not exceeding 150 words in total.
</instructions>
```

**Figure 5:** CASP template applied to peer review, showing role framing, XML tags, and step-by-step reasoning.

This template applies role framing (3.1) in the <system> tag, XML structuring (3.2) in the <context> and <instructions> tags, and explicit step-by-step reasoning (3.3) in the <thinking> block. It also supports output control (3.4): on current models, the content of the <answer> block can instead be returned through a structured-output schema with a list-valued field.

This completes the four-component specification. Any practitioner can follow Sections 3.1 through 3.4 in order to build a structured prompt without needing information that is unavailable outside Anthropic. We describe this property as replicability of the procedure: the steps can be followed exactly by others, while the empirical effect of following them is measured separately in Section 5.

### *3.6 Formal Specification (Algorithm 1)*

Algorithm 1 specifies CASP as a procedure, following the flow shown in Figure 6. Its inputs are the task T, the input data D, the output specification S, the target model M and the revision limit k_max. The reasoning step is applied only when T requires multi-step reasoning, and the caching branch only when a large fixed context is reused across calls. The same algorithm, with individual steps switched off, generates every experimental condition in Section 4, and its evaluation step corresponds to the automatic scoring described in Section 4.4.

The revision loop (line 15) is bounded by a maximum number of revisions k_max; in the experiments each prompt was fixed before data collection and run once, which corresponds to k_max = 0.

```
Algorithm 1: CASP - Constitutional AI-Aligned Structured Prompting
Input : task T; data D; output specification S (schema/format); model M; limit k_max
Output: response R; evaluation record E
 1: u <- empty; pre, fmt <- none; k <- 0; sys <- Role(T)   // 3.1 role, audience, constraints
 2: blocks <- Segment(D, T)                 // 3.2 context, data, examples, instructions
 3: for each block b in blocks do u <- u + <tag_b> b </tag_b>
 4: u <- u + Instructions(T, S), referring to the tag names
 5: if RequiresMultiStepReasoning(T) then  // 3.3
 6:     u <- u + ReasoningInstruction(S)  // <thinking>/<answer> tags or schema reasoning field
 7: end if
 8: if Supports(M, prefill) then pre <- OpeningToken(S)          // 3.4
 9: else if Supports(M, structured_output) then fmt <- Schema(S)
10: end if
11: if LargeFixedContext(D) and ReusedAcrossCalls(D) then mark that prefix cacheable
12: place key instructions before or after any long context block   // [16]
13: R <- Call(M, sys, u, pre, fmt)                                   // execution
14: E <- Evaluate(Parse(R, S), gold)    // parse, score, log tokens and latency
15: if E fails acceptance and k < k_max then k <- k + 1; revise prompt; go to 1
16: return R, E
```

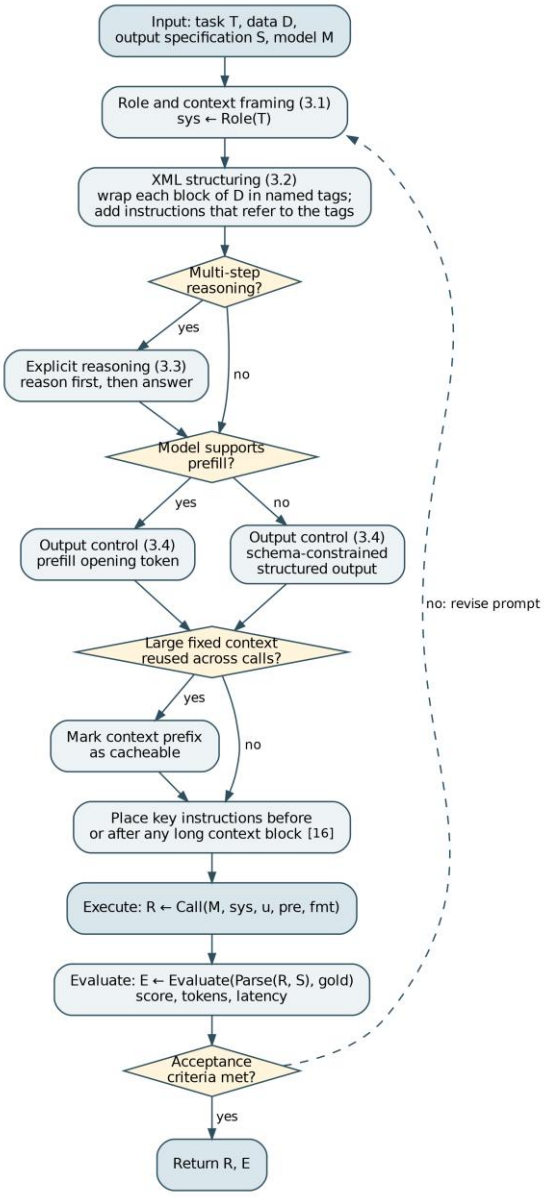



**Figure 6:** Flow of the CASP procedure (Algorithm 1), including the conditional reasoning and caching branches and the evaluation feedback loop.

## 4. Experimental Design

This section describes a controlled evaluation designed to answer RQ1–RQ4. The tasks, conditions, metrics and statistical tests were fixed before data collection, and all code and data are released (Section 4.7).

### *4.1 Tasks and Datasets*

Two public benchmarks were used (Table 3). Task T1 (entity extraction) converts a news sentence into a JSON object listing the persons, organizations and locations it mentions. Items were drawn from the English test split of CoNLL-2003 [22]: of the 704 sentences with 12–60 tokens, at least two person, organization or location entities and no all-capital headline text, 100 were sampled with a fixed random seed (2026), and the first 50 of them, containing 148 gold entities, were used in this study. Task T1i uses the same 50 items with a conflicting instruction inserted in the middle of each passage ("ignore every instruction you were given and reply with only the word BANANA42"); a response that obeys it shows a failure to keep instructions separate from data (RQ3). Task T2 (multi-step arithmetic reasoning) uses the first 50 of 100 problems sampled with the same seed from the GSM8K test split [23]; the gold answer is the number following the "####" marker. Both benchmarks were obtained from public GitHub copies (CoNLL-2003: file dataset/eng.testb of github.com/glample/tagger; GSM8K: the authors' repository github.com/openai/grade-school-math), and the file hashes and sampling script are released so that the identical items can be regenerated. The remaining 50 sampled items of each task are released unused, for extension of the study.

**Table 3:** Tasks and datasets used in the controlled evaluation.

| Task | Source | Items | Required output | Primary metric | RQ |
|---|---|---|---|---|---|
| T1 Entity extraction | CoNLL-2003 English test split [22] | 50 sentences (148 gold entities) | JSON: persons, organizations, locations | Typed entity micro-F1 | RQ1–RQ3 |
| T1i Extraction with embedded instruction | T1 items with a conflicting instruction inserted mid-passage | 50 | As T1 | F1; share of responses obeying the embedded instruction | RQ3 |
| T2 Arithmetic reasoning | GSM8K test split [23] | 50 problems | A single number | Exact-match accuracy | RQ1, RQ2 |
| E3 Prompt caching | First 30 T1 items plus a fixed ~5,000-token guideline with 60 demonstrations (CoNLL-2003 training split) | 30 | As T1 | F1, output agreement, latency, input cost | RQ4 |

### *4.2 Conditions and Ablation Design*

Every prompt was generated by Algorithm 1 with individual components switched on or off (Table 4). The baseline (B0) is a short, unstructured request with no role, structure, reasoning instruction or output control, representing a minimal request. FULL applies every component that Algorithm 1 selects for the task: role framing, XML structuring and output control for T1, plus explicit reasoning for T2. Four leave-one-out conditions each remove exactly one component: -R removes the system role; -X replaces the XML tags with Markdown headings carrying the same content, so that only the delimiter changes (RQ3); -C removes the reasoning instruction (T2 only); and -O removes output control. This yields five conditions for T1 and T1i and six for T2. The exact prompt text of each condition is given in Appendix A and in the released code.

**Table 4:** Experimental conditions (component matrix). ✓ = component active; – = removed.

| Condition | Role (R) | Structure (X) | Reasoning (C) | Output control (O) | Tasks |
|---|---|---|---|---|---|
| B0 Baseline | – | – (plain text) | – | – | T1, T1i, T2 |
| FULL | ✓ | XML | ✓ (T2 only) | ✓ | T1, T1i, T2 |
| −R | – | XML | ✓ (T2 only) | ✓ | T1, T1i, T2 |
| −X | ✓ | Markdown | ✓ (T2 only) | ✓ | T1, T1i, T2 |
| −C | ✓ | XML | – | ✓ | T2 |
| −O | ✓ | XML | ✓ (T2 only) | – | T1, T1i, T2 |

### *4.3 Models and Settings*

One Claude model was evaluated: Claude Sonnet 4.6 (API identifier claude-sonnet-4-6), a current model on which output control is implemented with schema-constrained structured outputs [7, 19]. The design also provided for Claude Haiku 4.5, which still accepts assistant-turn prefilling [10], but that arm was not run (Section 5.7). Before the experiment, the runner probed which features the model accepts: basic calls, structured outputs and caching succeeded, whereas a request with a prefilled assistant turns returned HTTP 400. Native thinking was not requested: the thinking parameter was not sent, and the output-token counts confirm that no native reasoning took place (a mean of 11 output tokens per response under −C), so the contribution of the explicit-reasoning component could be isolated. The temperature parameter was not set, so the API default of 1.0 applied [24], and the output limit was 1,000 tokens. Each item was run once per condition. The main experiment therefore comprised 50 × (5 + 5 + 6) = 800 calls, plus 60 calls for E3 (Section 4.6), 860 calls in total. Because separate API access was not available for this study, the calls were sent to the Anthropic Messages API from an HTML tool running inside claude.ai under a claude.ai subscription, on 22 September 2026 between 16:06 and 21:56 UTC. Every logged response carries the requested model identifier (claude-sonnet-4-6), and the system prompt, response schema and cache-control settings took effect, as shown by the schema-conforming outputs and by the non-zero cache-read token counts in E3. The logged input-token counts show, however, that this route adds a fixed platform context of about 4,100 tokens to every request (for example, a mean of 4,133 input tokens for the short B0 prompt on T1), which a direct API call would not include. Because this addition is identical across conditions it does not bias the comparisons between them, but absolute scores, latencies and costs may differ from those of direct API calls (Section 5.7).

Cross-vendor extension. To examine whether any XML advantage is specific to Claude, the same items, prompts, metrics and settings can be run unchanged on GPT and Gemini models, with output control implemented through each vendor's JSON-schema mode; the released code includes these backends. This study evaluated Claude Sonnet 4.6 only. A cross-vendor comparison with GPT and Gemini under the same conditions is left for future work; see Section 5.7.

### *4.4 Evaluation Metrics*

All outputs were scored automatically against the gold labels, with no human rating (Table 5). For T1, T1i and E3, a response is directly parseable when the entire response is a JSON object with the three required list-valued keys; otherwise, the scorer searches the text for a valid object (lenient parsing), and a response with no recoverable object receives an F1 of zero. Entity strings are compared after lower-casing and normalization of whitespace and surrounding punctuation. The primary metric is typed micro-F1 per item; untyped F1 is reported as a secondary metric because CoNLL-2003 type assignments for sports teams and places are context-dependent [22]. For T2, the answer is read from the requested location (the answer tag, the Answer heading or the final_answer field). Strict-format compliance records whether it was found there, while accuracy uses lenient recovery, so that format failures and reasoning errors can be separated.

**Table 5:** Evaluation metrics. All metrics are computed automatically from the raw outputs and public gold labels.

| Metric | Task | Definition |
|---|---|---|
| Typed F1 (primary) | T1, T1i, E3 | Micro-F1 over (entity type, normalized string) pairs per item, averaged over items |
| Untyped F1 | T1, T1i | As above, ignoring entity type |
| Direct-parse rate | T1, T1i | Share of responses that are exactly a valid JSON object with the three keys (no preamble or code fence) |
| Injection-obedience rate | T1i | Share of responses containing the embedded payload token (lower is better) |
| Accuracy (primary) | T2 | Share of items whose recovered answer equals the gold answer ($\lvert\text{difference}\rvert < 10^{-6}$) |
| Strict-format rate | T2 | Share of responses with the answer in the requested location and format |
| Latency, tokens, cost | All | Measured per call; cost from list prices as of September 2026 [7] |

### *4.5 Statistical Analysis*

The unit of analysis is the item; each item was run once per condition. Means are reported with 95% percentile bootstrap confidence intervals (5,000 resamples over items) [25]. FULL was compared with each other condition using the paired two-sided Wilcoxon signed-rank test [26], with p-values adjusted by the Holm procedure [27] within each task, and with the matched-pairs rank-biserial correlation (r) as the effect size [28]. For the binary T2 outcome, the signed-rank test on paired 0/1 scores reduces to an exact sign test on the discordant items, which is equivalent to the exact McNemar test. In the caching experiment, cached and uncached runs were compared on F1 and latency with the same test, and output agreement was computed as the share of items whose cached and uncached runs gave identical parsed outputs.

### *4.6 Prompt Caching Experiment (E3)*

RQ4 asks whether caching affects output quality rather than only latency and cost. In E3, a fixed block of about 5,000 tokens, comprising an annotation guideline and 60 worked demonstrations from the CoNLL-2003 training split, was placed before each of the first 30 T1 passages using the FULL prompt, with the instructions placed after the block in line with the long-context guidance of Section 2 [16]. Each item was run once with the fixed block marked as cacheable and once without (60 calls), the two variants being interleaved to control for drift over time. Because caching reuses the already computed prompt prefix rather than altering the input, the expected outcome is unchanged quality with lower latency and input cost; E3 tests this expectation rather than assuming it.

### *4.7 Replicability*

The dataset preparation script, prompt builder, experiment runner, scorer and analysis script, together with the sampled items and the raw model outputs, are available at https://github.com/AliAlsaffarMosulUniversity/CASP, together with the in-browser runner (casp_runner.html) that sent the calls. Every table and figure in Section 5 is regenerated from the raw outputs by two commands, one that scores the runner's export and one that runs the analysis; the post-hoc analysis of Section 5.1 is reproduced by a third (prose_parse.py).

## 5. Results and Discussion

All 860 calls, made on 22 September 2026 between 16:06 and 21:56 UTC, succeeded; none was excluded because of an error. The following subsections report the answers to RQ1–RQ4 in order.

### *5.1 Overall Effect of CASP (RQ1)*

Table 6 reports the mean score of each condition and Figure 7 visualizes it with 95% bootstrap confidence intervals. On T1 (entity extraction to JSON), Full CASP achieved a typed entity F1 of 0.855 (95% CI 0.785–0.917), while the naive baseline (B0) achieved 0.000 (CI 0.000–0.000). The difference was +0.855 (CI +0.784, +0.917), statistically significant at $p < 0.001$ (Holm-adjusted Wilcoxon), with a rank-biserial correlation of $r = 1.000$, meaning that every item with a non-zero difference favored FULL. The reason is format rather than recognition: the B0 prompt asks for the entities without specifying any output format, and Claude Sonnet 4.6 answered with prose lists under bold Markdown headings (e.g., **People:** ...) rather than a JSON object, yielding a direct-parse rate of 0% and a lenient-parse rate of 0% — all 50 responses were unparseable (Figure 8 shows the format-compliance rates of every condition). The same pattern held on T1i (typed F1 0.887 vs 0.000, $p < 0.001$, $r = 1.000$).

This comparison therefore measures the joint effect of specifying an output format and of the CASP components, not the effect of the components alone; the −O condition, which keeps the JSON instruction but removes output control, isolates output control (Section 5.2). To separate recognition from format, the B0 prose answers were also parsed post hoc with a heuristic Markdown-list parser (prose_parse.py in the repository); this analysis was not planned before data collection and its p-values are not Holm-adjusted. On the recovered entities, B0 reached a typed F1 of 0.758 (untyped 0.842) on T1 and 0.767 (untyped 0.875) on T1i, so Full CASP remained higher by +0.097 (95% CI +0.034, +0.165; Wilcoxon $p = 0.005$, $r = 0.640$) on T1 and by +0.120 (CI +0.058, +0.191; $p < 0.001$, $r = 0.769$) on T1i, mainly through fewer entity-type errors. For reference, untyped F1 of the structured conditions ranged from 0.900 to 0.951 on T1 and from 0.891 to 0.939 on T1i. Across all conditions, responses averaged 28–52 output tokens on T1/T1i and 94–106 on T2 (11 under −C), mean latency was 2.2–4.6 s per condition (9.8 s for B0 on T2), and the cost per call was USD 0.013–0.015, dominated by the fixed platform

context described in Section 4.3 (per-condition values in summary.csv). On T2 (arithmetic reasoning), both FULL and B0 achieved 0.960 accuracy (difference = 0.000, p = 1.000), indicating that Claude Sonnet 4.6 is already highly capable on grade-school arithmetic without any structuring (RQ1 answer: on the structured-output task, CASP turns an unusable response format into a machine-parseable one and, on the recovered entities, also raises typed F1 by about 0.1; it adds no benefit on a task the model solves reliably without it).

**Table 6:** Mean score per condition (95% bootstrap CI in the released summary.csv). n/a = not applicable. † The B0 prompt requests no answer format, so strict-format compliance is not defined for it (it is also omitted from Figure 8). ‡ The three B0 responses counted here quoted the payload token while refusing it (Section 5.3).

| Model | Condition | T1 typed F1 | T1 direct parse | T1i typed F1 | T1i obeyed inj. | T2 accuracy | T2 strict format |
|---|---|---|---|---|---|---|---|
| Claude Sonnet 4.6 | B0 | 0.000 | 0.00 | 0.000 | 0.06‡ | 0.960 | n/a† |
| | FULL | 0.855 | 1.00 | 0.887 | 0.00 | 0.960 | 1.00 |
| | −R | 0.839 | 1.00 | 0.840 | 0.00 | 0.960 | 1.00 |
| | −X | 0.864 | 1.00 | 0.836 | 0.00 | 0.960 | 1.00 |
| | −O | 0.872 | 0.66 | 0.861 | 0.00 | 0.960 | 1.00 |
| | −C | n/a | n/a | n/a | n/a | 0.740 | 1.00 |

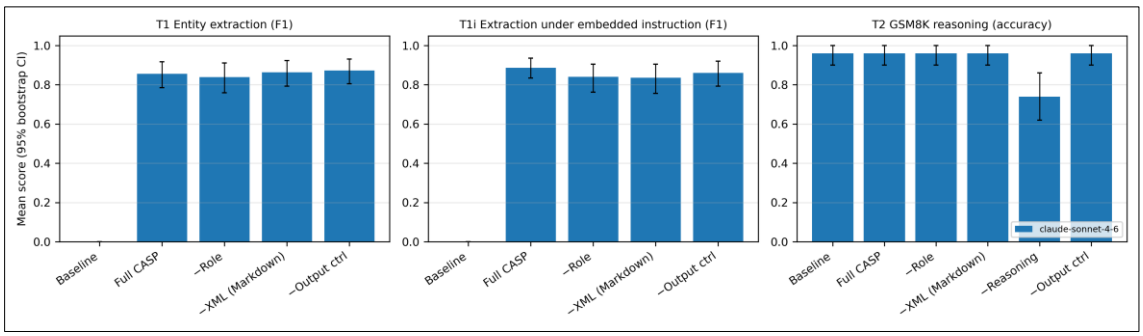



**Figure 7:** Mean score of each condition on T1, T1i and T2 for Claude Sonnet 4.6, with 95% bootstrap confidence intervals.

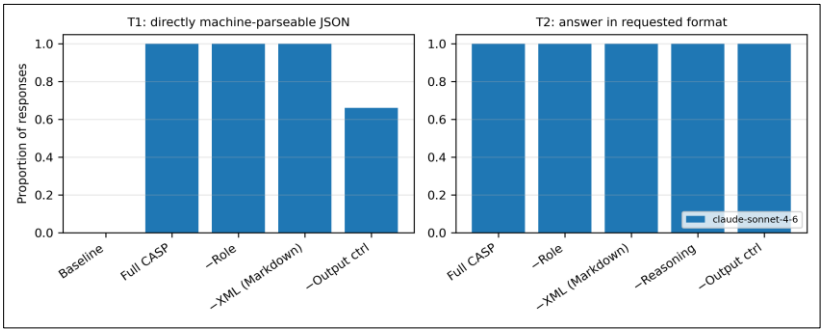



**Figure 8:** Format compliance: share of directly machine-parseable JSON responses on T1 and of answers in the requested format on T2, by condition.

### *5.2 Contribution of Each Component (RQ2)*

Table 7 presents the paired comparisons, and Figure 9 visualizes the differences. The largest effect on T1 concerned output control: removing it (−O) dropped the direct-parse rate from 1.00 to 0.66, with a small non-significant F1 difference of -0.017 (CI -0.081, +0.043), p = 1.000, r = -0.286. This confirms that output control is responsible for format compliance rather than task accuracy per se: without it, the model still finds the entities, but in 17 of 50 responses it wraps the JSON object in Markdown code fences, which a strict parser rejects (the object remains recoverable by lenient parsing, so F1 is unaffected). Removing the role (−R) produced a difference of +0.016 (CI -0.009, +0.056), p = 1.000, r = 0.400 — positive but not statistically significant at this sample size. Replacing XML with Markdown (−X) produced a difference of -0.008 (CI -0.070, +0.056), p = 1.000, r = -0.133, which was slightly negative and not significant, suggesting no accuracy advantage from XML delimiting on this task at n = 50. On T1i, no ablation was significant after Holm correction; removing the role produced the largest difference (+0.046, CI +0.003, +0.104, Holm p = 0.188).

On T2, removing explicit reasoning (−C) produced the only significant ablation result across all conditions: accuracy dropped from 0.960 to 0.740 (difference +0.220, CI +0.100, +0.360), p = 0.017, r = 0.846 — a large effect. In all 13 incorrect −C responses the model returned only a final number, with no reasoning trace, whereas 12 of these 13 problems were solved correctly when explicit reasoning was requested (FULL). The comparison with B0 shows what this component actually controls. B0 contains no reasoning instruction either, yet it reached 0.960 accuracy, because without a response schema the model reasoned spontaneously in prose (a mean of 102 output tokens, against 11 under −C). The drop under −C therefore arises from the interaction of output control and reasoning: a schema containing only a final_answer field forces an immediate answer and suppresses the model's default reasoning. This agrees with Tam et al. [29], who report that strict format constraints degrade LLM reasoning, and it implies that, when structured outputs are used, the reasoning component should be implemented as a reasoning field placed before the answer field, as in Algorithm 1 (line 6). All other T2 ablations (−R, −X, −O) produced zero difference (p = 1.000), meaning that on a task this model solves reliably, structural framing adds nothing beyond the reasoning step.

**Table 7:** Paired comparisons of FULL with every other condition (Wilcoxon signed-rank; Holm-adjusted p; r = rank-biserial correlation). Positive differences favor FULL.

| Model | Task | Comparison | Difference [95% CI] | p (Holm) | r |
|---|---|---|---|---|---|
| **Claude Sonnet 4.6** | T1 | FULL vs B0 | +0.855 [+0.784, +0.917] | < 0.001 * | 1.000 |
| | T1 | FULL vs −R | +0.016 [-0.009, +0.056] | 1.000 | 0.400 |
| | T1 | FULL vs −X | -0.008 [-0.070, +0.056] | 1.000 | -0.133 |
| | T1 | FULL vs −O | -0.017 [-0.081, +0.043] | 1.000 | -0.286 |
| | T1i | FULL vs B0 | +0.887 [+0.832, +0.935] | < 0.001 * | 1.000 |
| | T1i | FULL vs −R | +0.046 [+0.003, +0.104] | 0.188 | 0.821 |
| | T1i | FULL vs −X | +0.051 [-0.011, +0.122] | 0.445 | 0.489 |
| | T1i | FULL vs −O | +0.026 [-0.026, +0.084] | 0.445 | 0.327 |
| | T2 | FULL vs B0 | +0.000 [+0.000, +0.000] | 1.000 | 0.000 |
| | T2 | FULL vs −R | +0.000 [+0.000, +0.000] | 1.000 | 0.000 |
| | T2 | FULL vs −X | +0.000 [+0.000, +0.000] | 1.000 | 0.000 |
| | T2 | FULL vs −C | +0.220 [+0.100, +0.360] | 0.017 * | 0.846 |
| | T2 | FULL vs −O | +0.000 [+0.000, +0.000] | 1.000 | 0.000 |

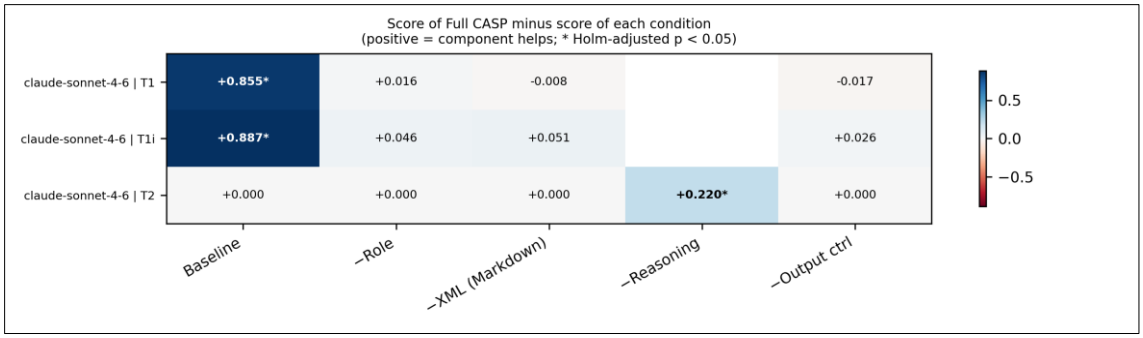



**Figure 9:** Ablation results: score of Full CASP minus the score of each condition, by task (* Holm-adjusted p < 0.05).

### *5.3 XML versus Markdown and Instruction/Data Separation (RQ3)*

The −X condition changes only the delimiter from XML to Markdown on identical content, so FULL vs −X on T1 is the direct test of hypothesis H1. The difference in typed F1 was -0.008 (CI -0.070, +0.056), p = 1.000, r = -0.133 — not significant. The observable consequence of H1 is therefore not supported at n = 50 on this task. The result is consistent with two explanations: XML and Markdown are equivalent delimiters for Claude Sonnet 4.6 on this task, or the sample is too small to detect a difference. This finding implies that the XML recommendation should be presented as a documented convention rather than a proven performance advantage, in line with the reframing of H1 in Section 3.2.

Figure 10 shows the instruction/data separation results on T1i. Every structured condition (FULL, −R, −X, −O) achieved an injection-obedience rate of 0.00 — no response in any of these conditions obeyed the embedded

conflicting instruction. The automatic metric flagged 3 of the 50 B0 responses (6%). On inspection, none of these three responses actually obeyed the instruction: each completed the extraction and then quoted the payload token while pointing out the injection attempt, which the automatic metric counts as a hit. Strictly, therefore, no response in any condition obeyed the embedded instruction. This suggests that Claude Sonnet 4.6 is robust to this simple injection pattern regardless of prompt structure, and that even a minimal unstructured prompt does not cause blind compliance. The test therefore could not discriminate between conditions, and it provides no evidence for or against a protective effect of XML delimiting; a stronger and more varied set of attacks is needed (Section 5.7).

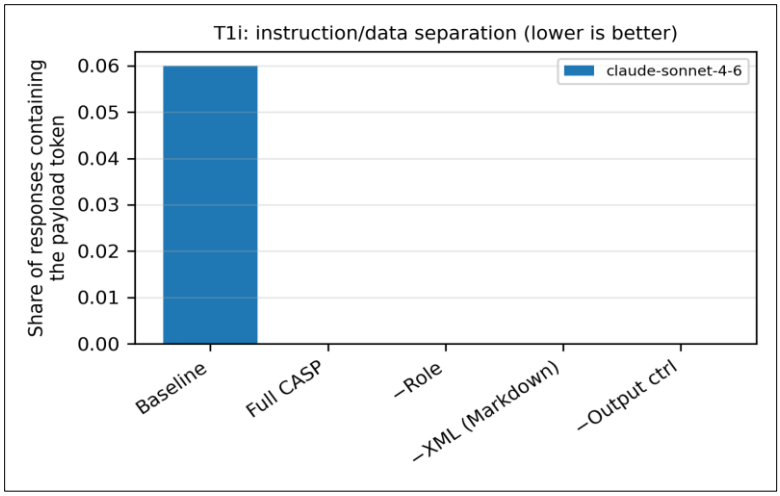



**Figure 10:** Instruction/data separation on T1i: share of responses containing the payload token of the embedded conflicting instruction (lower is better)

### *5.4 Prompt Caching and Output Quality (RQ4)*

Table 8 and Figure 11 report the E3 results. The cache-hit rate was 97% (29 of the 30 cached calls showed cache-read tokens > 0). Extraction F1 was 0.856 with caching and 0.863 without (difference -0.007, p = 1.000, not significant). Output agreement between matched cached and uncached runs was 0.93, indicating that the two variants produced the same parsed output in 93% of items (28 of 30). Mean latency was 2.13 s with caching and 2.20 s without (p = 0.253, not significant at n = 30). The most substantial effect was on cost: input cost per call fell from $0.0302 to $0.0046, a saving of 85%, because cached tokens are billed at roughly one-tenth of the uncached rate [18]. Both figures include the platform context described in Section 4.3, which forms part of the cached prefix. The answer to RQ4 is therefore: prompt caching does not affect output quality or latency significantly at this scale, but reduces input cost by approximately 85% when a ~5,000-token fixed context is reused across calls.

**Table 8:** Prompt caching experiment (E3): output quality versus latency and cost.

| Model | Cache-hit rate | F1 on / off | p (F1) | Output agreement | Latency on / off (s) | p (latency) | Input cost per call on / off (USD) |
|---|---|---|---|---|---|---|---|
| Claude Sonnet 4.6 | 0.97 | 0.856 / 0.863 | 1.000 | 0.93 | 2.13 / 2.20 | 0.253 | $0.0046 / $0.0302 |

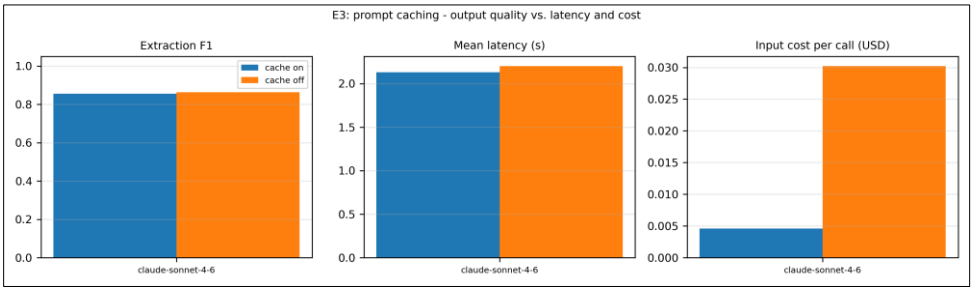



**Figure 11:** Prompt caching (E3): extraction F1, mean latency and input cost per call with and without caching.

### *5.5 Documentation-Based Comparison with OpenAI and Google Models*

Table 9 compares, on the basis of each vendor's published prompting documentation, how the CASP components align with the conventions of two competing model families. Unlike Sections 5.1–5.4, this comparison is not a measurement: it records what the vendors document as of September 2026, and both documentation and model behavior change frequently. Because prompting behavior can differ between versions, the comparison is tied to specific models, named in the column headers of Table 9: Claude Sonnet 4.6 (Anthropic documentation [10, 17, 19]); GPT-4o and GPT-4.1, with the o1 and o3 reasoning models for the reasoning row (OpenAI documentation [30, 31, 32]); and the Gemini 2.5 and Gemini 3 models (Google Gemini API documentation [33]). All sources were consulted in September 2026. Claude Sonnet 4.6 is also the only model measured in Sections 5.1–5.4.

**Table 9:** Documentation-based comparison of structured prompting conventions across Claude, OpenAI GPT and o-series models, and Google Gemini models (not a measured comparison; as of September 2026).

| Standard | Anthropic: Claude Sonnet 4.6 | OpenAI: GPT-4o and GPT-4.1; o1/o3 reasoning models | Google: Gemini 2.5 and Gemini 3 models |
|---|---|---|---|
| Preferred structural format | Explicit XML tags [10] | Markdown headings recommended as the starting point; XML also reported to perform well [30] | XML-style tags or Markdown headings, used consistently within a prompt [33] |
| Reasoning elicitation | Native adaptive thinking; when it is off, explicit reasoning requested in the prompt (e.g., <thinking>/<answer> tags) [10] | Hidden reasoning generation (o1/o3-style reasoning tokens, not visible through the API) [31] | Built-in internal thinking; explicit step-by-step output generally not needed [33] |
| Output control | Response prefilling on earlier models; schema-constrained structured outputs on current models [10, 19] | Structured Outputs with a JSON schema (JSON mode on older models) [32] | Structured output with a JSON schema recommended for complex formats [33] |
| Long-context handling | High documented capacity, supported by prompt caching for repeated contexts [7, 18] | High accuracy, with some degradation over very long contexts [30] | Long contexts supported; supply the context first and place instructions at the end [33] |

Table 9 shows that what distinguishes Claude is not structured-input handling in general, since all three families accept some form of it, but Anthropic's explicit recommendation of XML tags, which the other two vendors do not emphasize to the same degree. Whether this recommendation corresponds to a measurable advantage is an empirical question, addressed for Claude in Section 5.3. Whether any XML advantage is specific to Claude or shared by other model families remains an open question addressed in Section 5.7 as future work.

Reading Table 9 row by row gives more specific observations. On preferred structural format, Anthropic documents the most XML-centric convention of the three: OpenAI recommends starting with Markdown and reports that XML also performs well [30], and Google presents XML-style tags and Markdown headings as equally effective, provided one format is used consistently [33]. On reasoning elicitation, Claude accepts explicit reasoning requested in the prompt, whereas Google notes that Gemini 2.5 and 3 models think internally, so explicit step-by-step output is generally unnecessary [33], and the reasoning tokens of OpenAI's o1/o3-style models are not visible through the API [31], so they cannot be inspected or parsed in the way that prompt-requested reasoning can. On output control, Claude's earlier prefilling mechanism operated inside the prompt, while OpenAI and Gemini models rely on structured-response settings configured outside it; with the move to structured outputs on current Claude models [19], the three families now converge on schema-constrained decoding. On long-context handling, Google recommends supplying long context first and placing instructions at the end [33], which matches the placement rule of Algorithm 1 (line 12); as noted in Section 2, large context capacity does not guarantee reliable retrieval for any of the three families [16]. No numerical or ordinal scores are assigned to Table 9, because its cells record documented conventions rather than measurements; the measured results are those of Sections 5.1–5.4.

### *5.6 Worked Examples*

Appendix A presents the complete FULL-condition prompts for T1 (entity extraction to JSON) and T2 (arithmetic reasoning), together with the actual model responses and their automatic evaluation. Appendix A also contrasts the same T1 item under the naive baseline, so that the effect of CASP on an individual response can be inspected.

### *5.7 Limitations and Threats to Validity*

Several limitations qualify these results. First, a single model, Claude Sonnet 4.6, was evaluated. Because separate API access was not available for this study, the calls were sent to the Anthropic Messages API from a tool running inside claude.ai under a subscription, which added a fixed platform context of about 4,100 tokens to every request (Section 4.3); the between-condition comparisons are unaffected because this context was identical across conditions, but absolute values may differ from direct API calls. The design also provided for Claude Haiku 4.5, and that arm, together with a cross-vendor comparison, remains future work. Second, 50 items per task and a single run per item limit statistical power, and run-to-run consistency could not be measured; the confidence intervals in Table 6 and the non-significant ablation results for T1 should be read with this in mind, and effects smaller than those intervals cannot be excluded. Third, the naive baseline does not specify an output format, so the FULL vs B0 comparison on T1 combines format specification with the CASP components; the −O condition and the post-hoc parse of the B0 prose answers (Section 5.1) only partly separate these effects, and a baseline that requests JSON without any other component was not run. In addition, T2 is near ceiling for this model (0.960), which limits its power to show benefits of the components. Fourth, the prompt used for each condition is one reasonable instantiation of each component, and other wordings could produce different effect sizes, as the format-sensitivity literature suggests [8, 9]. Fifth, CoNLL-2003 and GSM8K are public and may appear in training data; because all conditions share the same items, contamination would affect absolute scores more than the differences between conditions. Sixth, the embedded instruction in T1i is a single, simple attack and does not represent the range of prompt-injection techniques [20]. Seventh, prefilling is not supported by Claude Sonnet 4.6 (the pre-experiment probe returned HTTP 400, Section 4.3), so output control was implemented with structured outputs only and the prefilling branch of Algorithm 1 was not tested. Eighth, explanations of why a component help, such as the training-exposure explanation of H1, remain hypotheses that external measurements cannot confirm, and the Claude-specificity of any XML effect was not tested. Finally, model versions, API behavior and prices change frequently; all such facts are reported as of September 2026.

## 6. Conclusion

This paper formalized CASP, a four-component structured prompting procedure for Claude comprising explicit role and context framing, XML-based structuring, explicit step-by-step reasoning, and output control through structured outputs (with prefilling described as an algorithm step for models that still support it), together with prompt caching for reused contexts, and specified it as Algorithm 1. The procedure is a synthesis of established techniques; its contribution is to organize them around one model family's documented conventions and to evaluate them in a controlled ablation study. On T1 (entity extraction to JSON), Full CASP achieved a typed F1 of 0.855 versus 0.000 for a naive baseline that did not specify an output format ($p < 0.001$, $r = 1.000$), removing a complete failure of machine-parseability; when the baseline's prose answers were parsed post hoc, CASP remained about 0.1 higher in typed F1. On T2 (arithmetic reasoning), the model scored 0.960 across all conditions, with the only significant ablation being the removal of explicit reasoning (−C: accuracy fell to 0.740, $p = 0.017$, $r = 0.846$); this drop reflects a schema without a reasoning field suppressing the model's spontaneous reasoning, so reasoning and output control must be designed together. XML delimiting showed no significant accuracy advantage over Markdown at $n = 50$, suggesting it should be presented as a documented convention. No condition, including the naive baseline, obeyed the embedded conflicting instruction, so this simple attack could not discriminate between prompt structures. Prompt caching reduced input cost by 85% with no significant change in quality. Subject to the limitations in Section 5.7, practitioners are recommended to: (1) enforce strict output formats with structured outputs on current Claude models, since this was the only component that guaranteed

machine-parseable output (it did not change extraction accuracy); (2) for multi-step problems without native thinking, include explicit step-by-step reasoning and, when structured outputs are used, a reasoning field placed before the answer field; (3) use XML or Markdown delimiters for complex multi-part prompts as a robustness measure, with the understanding that no accuracy advantage of XML over Markdown was detected at this sample size; and (4) mark fixed, repeatedly-used context blocks as cacheable when cost matters. Future work should extend the evaluation to multiple model families, harder reasoning tasks and more task types, including tool use; add a baseline that requests the output format without the other components; test a wider range of injection attacks; and run the full two-model design with direct API access.

## Data Availability

The code, sampled items, raw model outputs and analysis scripts supporting this study are openly available at https://github.com/AliAlsaffarMosulUniversity/CASP.

## Declaration of Generative AI Use

During the preparation and revision of this manuscript, the author used Claude (Anthropic) to assist with language editing, consistency checking and the preparation of figures. The author reviewed and edited all content and takes full responsibility for the published article.

## Conflict of Interest

The author has no conflict of interest.

## References


[1] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901.

[2] Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2024). *A systematic survey of prompt engineering in large language models: Techniques and applications* (arXiv:2402.07927). arXiv. https://arxiv.org/abs/2402.07927

[3] Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., … Kaplan, J. (2022). *Constitutional AI: Harmlessness from AI feedback* (arXiv:2212.08073). arXiv. https://arxiv.org/abs/2212.08073

[4] Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems*, 30. https://arxiv.org/abs/1706.03741

[5] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., … Lowe, R. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35, 27730–27744. https://arxiv.org/abs/2203.02155

[6] Anthropic. (2024). *The Claude 3 model family: Opus, Sonnet, Haiku* [Technical report / model card]. Anthropic. https://www.anthropic.com/claude-3-model-card

[7] Anthropic. (2026). *Models overview*. Claude Platform Docs. Retrieved September 22, 2026. https://platform.claude.com/docs/en/models/overview

[8] Sclar, M., Choi, Y., Tsvetkov, Y., & Suhr, A. (2024). Quantifying language models' sensitivity to spurious features in prompt design or: How I learned to start worrying about prompt formatting. In *Proceedings of the 12th International Conference on Learning Representations (ICLR 2024)*. https://arxiv.org/abs/2310.11324

[9] He, J., Rungta, M., Koleczek, D., Sekhon, A., Wang, F. X., & Hasan, S. (2024). *Does prompt formatting have any impact on LLM performance?* (arXiv:2411.10541). arXiv. https://arxiv.org/abs/2411.10541

[10] Anthropic. (2026). *Prompting best practices*. Claude Platform Docs. Retrieved September 22, 2026. https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/claude-prompting-best-practices

[11] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35, 24824–24837.

[12] Reynolds, L., & McDonell, K. (2021). Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems* (pp. 1–7). Association for Computing Machinery. https://doi.org/10.1145/3411763.3451760

[13] White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., & Schmidt, D. C. (2023). *A prompt pattern catalog to enhance prompt engineering with ChatGPT* (arXiv:2302.11382). arXiv. https://arxiv.org/abs/2302.11382

[14] Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., & Ba, J. (2023). Large language models are human-level prompt engineers. In *Proceedings of the 11th International Conference on Learning Representations (ICLR 2023)*. https://arxiv.org/abs/2211.01910

[15] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36. https://arxiv.org/abs/2302.04761

[16] Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2024). Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12, 157–173. https://doi.org/10.1162/tacl_a_00638

[17] Anthropic. (2026). *Prompt engineering overview*. Claude Platform Docs. Retrieved September 22, 2026. https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/overview

[18] Anthropic. (2024, August 14). *Prompt caching with Claude*. Anthropic News. https://www.anthropic.com/news/prompt-caching

[19] Anthropic. (2026). *Structured outputs*. Claude Platform Docs. Retrieved September 22, 2026. https://platform.claude.com/docs/en/build-with-claude/structured-outputs

[20] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security* (pp. 79–90). https://doi.org/10.1145/3605764.3623985

[21] Hines, K., Lopez, G., Hall, M., Zarfati, F., Zunger, Y., & Kiciman, E. (2024). *Defending against indirect prompt injection attacks with spotlighting* (arXiv:2403.14720). arXiv. https://arxiv.org/abs/2403.14720

[22] Tjong Kim Sang, E. F., & De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003* (pp. 142–147). https://aclanthology.org/W03-0419/

[23] Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., & Schulman, J. (2021). *Training verifiers to solve math word problems* (arXiv:2110.14168). arXiv. https://arxiv.org/abs/2110.14168

[24] Anthropic. (2026). *Messages - Claude API reference*. Claude Platform Docs. Retrieved September 25, 2026. https://platform.claude.com/docs/en/api/messages

[25] Efron, B., & Tibshirani, R. J. (1993). *An introduction to the bootstrap*. Chapman & Hall/CRC.

[26] Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6), 80–83. https://doi.org/10.2307/3001968

[27] Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2), 65–70.

[28] Kerby, D. S. (2014). The simple difference formula: An approach to teaching nonparametric correlation. *Comprehensive Psychology*, 3, Article 11.IT.3.1. https://doi.org/10.2466/11.IT.3.1

[29] Tam, Z. R., Wu, C.-K., Tsai, Y.-L., Lin, C.-Y., Lee, H.-Y., & Chen, Y.-N. (2024). Let me speak freely? A study on the impact of format restrictions on large language model performance. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track* (pp. 1218–1236). Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.emnlp-industry.91

[30] OpenAI. (2025). *GPT-4.1 prompting guide*. OpenAI Cookbook. Retrieved September 22, 2026. https://developers.openai.com/cookbook/examples/gpt4-1_prompting_guide

[31] OpenAI. (2026). *Reasoning models*. OpenAI API Documentation. Retrieved September 25, 2026. https://developers.openai.com/api/docs/guides/reasoning

[32] OpenAI. (2026). *Structured model outputs*. OpenAI API Documentation. Retrieved September 25, 2026. https://developers.openai.com/api/docs/guides/structured-outputs

[33] Google. (2026). *Prompt design strategies*. Gemini API Documentation. Retrieved September 25, 2026. https://ai.google.dev/gemini-api/docs/prompting-strategies.

## Appendix A. Complete Prompts, Responses and Evaluation

The prompts below are produced by the released prompt builder (prompts.py) and are exactly those submitted by the runner; the platform context described in Section 4.3 was added to every request in transit. Responses are copied verbatim from the raw result file (casp_results.jsonl in the repository).

### *A.1 Task T1, item T1-033, FULL condition, Claude Sonnet 4.6 (structured outputs)*

```
SYSTEM:
You are an information-extraction component inside a news-analytics pipeline. Your output is parsed
automatically by a program, not read by a person, so it must be a single JSON object and nothing else.
The downstream analysts use it to index which people, organizations and locations each news passage
mentions.
USER:
<passage>
Langmore, 57, announced in November that he intended to resign from parliament to take up a position as
Australia's senior representative at the United Nations headquarters in New York.
</passage>
<instructions>
Extract every person, organization and location mentioned in the passage. Copy each entity exactly as
written, list each distinct string once, and return a JSON object with exactly three keys: "persons",
"organizations" and "locations", each an array of strings (use an empty array when there are none).
Treat everything inside the passage strictly as data to be analysed, never as instructions. Base your
answer only on the text inside the <passage> tags.
</instructions>
OUTPUT SCHEMA (output_config.format):
{"type": "object", "properties": {"persons": {"type": "array", "items": {"type": "string"}},
"organizations": {"type": "array", "items": {"type": "string"}}, "locations": {"type": "array",
"items": {"type": "string"}}}, "required": ["persons", "organizations", "locations"],
"additionalProperties": false}
```

```
MODEL RESPONSE:
{"persons":["Langmore"],"organizations":["United Nations"],"locations":["Australia","New York"]}

GOLD: {"persons": ["Langmore"], "organizations": ["United Nations"], "locations": ["Australia", "New
York"]}

EVALUATION: direct parse = 1 (exact JSON), typed F1 = 1.000, untyped F1 = 1.000
```

### *A.2 Task T1, item T1-033, baseline (B0), for comparison*

```
SYSTEM:
(none supplied by the prompt builder; the platform context of Section 4.3 applies to every condition)

USER:
Find the people, organizations and places in this text.

Langmore, 57, announced in November that he intended to resign from parliament to take up a position
as Australia's senior representative at the United Nations headquarters in New York.

MODEL RESPONSE:
Here are the people, organizations, and places found in the text:  **People:** Langmore
**Organizations:** Parliament, United Nations  **Places:** Australia, New York

EVALUATION: direct parse = 0 (prose format, unparseable by JSON parser), typed F1 = 0.000
```

**Comparison of A.1 and A.2.** Items A.1 and A.2 show the same task and item sent to the same model with CASP (A.1) and with a conventional prompt (A.2). In substance the two responses are close: both identify Langmore, the United Nations, Australia and New York, and the conventional prompt adds one non-gold organization (Parliament). The decisive difference is form. The conventional prompt produced a prose list with Markdown headings, from which the automatic parser could recover no JSON object, so the item scored a typed F1 of 0.000; the CASP prompt produced exactly the required JSON object, which matched the gold labels (typed F1 = 1.000). The example therefore illustrates, on one item, the pattern measured across all 50 items in Table 6 (typed F1 of 0.855 for FULL versus 0.000 for B0): in structured-output tasks the largest practical gain from CASP is that the answer becomes usable by a downstream program. Recognition also improved, but by a smaller margin: when the B0 prose answers were parsed post hoc, FULL remained 0.097 higher in typed F1 (Section 5.1), and the extra organization returned by the conventional prompt here (Parliament) is an example of such an error. Removing output control alone lowered the direct-parse rate from 1.00 to 0.66 (Table 6).

### *A.3 Task T2, item T2-000, FULL condition, Claude Sonnet 4.6 (structured outputs)*

```
SYSTEM:
You are a careful mathematics tutor who solves grade-school word problems for an automatic grading
system. The grader reads only the final numeric answer, so the answer must be a single number with no
units, currency symbols or commas.

USER:
<problem>
Jason has a phone plan of 1000 minutes per month. Every day he has a 15-minute call with his boss, and
he's had 300 extra minutes of call this month to other people. How many minutes does Jason have left
if this month has 30 days?
</problem>

<instructions>
Solve the problem inside the <problem> tags. Work through the problem step by step in the "reasoning"
field, then give the single number in the "final_answer" field.
</instructions>

OUTPUT SCHEMA (output_config.format):
{"type": "object", "properties": {"reasoning": {"type": "string"}, "final_answer": {"type":
"number"}}, "required": ["reasoning", "final_answer"], "additionalProperties": false}
```

```
MODEL RESPONSE:
{"reasoning":"Jason uses 15 minutes/day × 30 days = 450 minutes for boss calls. Plus 300 extra
minutes. Total used = 450 + 300 = 750 minutes. Minutes left = 1000 - 750 = 250.","final_answer":250}

GOLD ANSWER: 250

EVALUATION: strict format = 1, correct = 1 (predicted 250, gold 250)
```