RT-Seg: A Toolkit for Reasoning Trace Segmentation

Leon Hammerla, Bhuvanesh Verma, Alexander Mehler
Goethe University Frankfurt am Main
{hammerla,verma,mehler}@em.uni-frankfurt.de

Software & Documentation: https://github.com/texttechnologylab/RT-SEG
Software Archive: https://doi.org/10.5281/zenodo.21416283
Video Demonstration: https://www.youtube.com/watch?v=5RyLA3qElbw

Abstract

Reasoning traces are increasingly used to analyze and evaluate large language models (LLMs), yet they are often treated as monolithic text or segmented using heuristics, such as sentence boundaries. However, reasoning traces admit multiple semantically or epistemically motivated decompositions that may not align with heuristically segmented units and that can vary in granularity depending on the underlying annotation scheme. We introduce RT-Seg, a modular toolkit for reasoning trace segmentation that supports diverse segmentation engines, fusion strategies, and labeling schemas within a single framework. RT-Seg allows researchers to flexibly define, combine, and compare alternative reasoning-step formulations, enabling structured, fine-grained analyses of model reasoning behavior. We evaluate RT-Seg using manually annotated traces and two annotation schemes. Our exploratory evaluation shows that heterogeneous ensemble configurations can achieve higher boundary similarity than single-engine approaches and, under the searched configurations, approach observed inter-human reference scores. We provide RT-Seg as an open-source Python library with a Docker-based text user interface, together with newly annotated reasoning traces, to enable reproducible and exploratory analysis of reasoning traces.

1 Introduction

Large language models (LLMs) increasingly produce reasoning traces: extended sequences of intermediate steps that have been shown to substantially improve model performance (Wei et al., 2022) while also providing unprecedented visibility into model behavior (Yu et al., 2025; Wei Jie et al., 2024). Such traces appear in a wide range of settings, including large reasoning models (LRMs) such as DeepSeek-R1 (Guo et al., 2025) and Qwen3 (Yang et al., 2025), chain-of-thought prompting (CoT) (Wei et al., 2022), tool-augmented reasoning (Song et al., 2025; Yao et al., 2023; Schick et al., 2023), multi-agent systems for reasoning (Lei et al., 2025; Du et al., 2024), and post-hoc explanations (Turpin et al., 2023). Despite their importance, reasoning traces are typically treated as monolithic text, obscuring internal structure (Lightman et al., 2024; Uesato et al., 2022), or segmented using simple heuristics such as sentence or line boundaries (e.g. Bogdan et al., 2025; Nguyen et al., 2024; Golovneva et al., 2023). From a linguistic point of view, however, a trace consists of multiple reasoning steps (e.g., decomposition (Zhou et al., 2023), inference, verification, or revision (Madaan et al., 2023)) that unfold sequentially (Zhou et al., 2024) and whose boundaries are often implicit and may vary across models, prompts, concepts, and tasks (Korbak et al., 2025; Xiong et al., 2025; Zheng et al., 2024). Different segmentation schemes introduced in (Lee et al., 2025; Bogdan et al., 2025; Hammoud et al., 2025) can yield different interpretations of the same trace, complicating step-level analysis and structured comparison.

Thus, while reasoning-trace analysis is becoming an increasingly important task, systems offering the interpretive flexibility required to explore this phenomenon remain lacking. To fill this gap, we present **RT-Seg**, a practical and extensible toolkit for reasoning trace segmentation. RT-Seg provides a modular framework for (1) defining, (2) combining, and (3) comparing segmentation and labeling strategies, enabling structured inspection of reasoning traces across models and tasks. RT-Seg is designed for ease of deployment and integration into existing research workflows. It is accessible both as a Python library and as a Dockerized application with an interactive text user interface (TUI). This dual interface supports large-scale experiments as well as exploratory, human-in-the-loop trace analysis (see Figure 1). Our contributions are threefold:

1. We formalize reasoning-trace segmentation as a flexible, first-class task, explicitly supporting multiple plausible step definitions instead of committing to a single heuristic granularity.

2. We introduce RT-Seg, a modular framework that integrates heterogeneous segmentation engines, fusion strategies, and labeling schemas, enabling systematic comparison and composition of reasoning-step formulations.

3. We release RT-Seg as an open-source Python library with a Dockerized TUI, together with a dataset of 100 reasoning traces annotated under the Thought Anchor schema by two independent human annotators, comprising more than 2,500 labeled reasoning steps per annotator.[1] Together, these resources support reproducible and extensible step-level reasoning-trace analysis.

2 RT-Seg

2.1 Architecture and Overview

RT-Seg is designed to integrate and aggregate heterogeneous segmentation and labeling approaches for reasoning traces. Its architecture therefore follows a modular, ensemble-based design. At its core, RT-Seg maintains an ensemble of segmentation engines (see Table 2 for the list of the 19 integrated engines and subsection 2.2 for an in-depth description of their types). Each engine independently produces (1) a segmentation of a given reasoning trace and optionally (2) a labeling of the resulting segments according to its underlying conceptual framework (e.g., linguistic, semantic, epistemic). When multiple segmentation engines are used, RT-Seg reconciles their outputs through late-fusion aligners (five are currently implemented; see Figure 1 and subsection 2.3). Aligners merge alternative segmentations into a single unified segmentation. Supported strategies include, in particular, union-based, intersection-based, and fuzzy alignment schemes, allowing users to control the granularity and strictness of agreement across engines. Once a unified segmentation is obtained, RT-Seg aggregates segment-level labels across engines via two mechanisms: (1) *concatenation*, where a segment inherits all labels from overlapping segments across engines, and (2) *majority voting*, which is particularly suitable when engines share a common labeling schema. RT-Seg operates over configurable base units and supports sentence-level (coarse) and clause-level (fine-grained) segmentation. By decoupling segmentation, alignment, and label aggregation, RT-Seg enables flexible composition of heterogeneous reasoning-step definitions while preserving comparability across schemes. RT-Seg is implemented in Python with an extensible interface design. New segmentation engines and aligners can be integrated by inheriting from the provided base classes and implementing a single generic method with fixed input–output specifications. For components that rely on LLMs, custom system and user prompts are supported and can be configured globally via a JSON configuration file or locally within Python code[2]. For exploratory and small-scale usage, RT-Seg provides a containerized application exposing a terminal user interface (TUI). It lets users input reasoning traces, select combinations of segmentation engines, aligners, and granularity settings, and interactively inspect the resulting segmentations.

2.2 Segmentation Engines

RT-Seg integrates 19 segmentation engines. Each of these engines is implemented as a Python class inheriting from a shared base class. An engine must implement the `_segment(trace:str,**kwargs)` method at minimum, which takes a reasoning trace as input along with optional keyword arguments and returns a segmentation of the trace as a list of character offset pairs.[3]

2.2.1 Heuristic Engines

We implement a family of heuristic segmentation engines that rely on surface-level structural and discourse cues. These engines target common generation artifacts observed in model reasoning traces. In particular, the **Line-Break Engine** inserts boundaries at structural markers such as double line breaks, which often correspond to implicit step transitions, while the **Regex Engine** uses predefined discourse markers (e.g., *wait*, *let*, *so*, *however*) to detect likely epistemic shifts within traces.

[1] Repository with code, dataset, Docker setup, and demo video: https://github.com/texttechnologylab/RT-SEG

[2] Extensive documentation is available on GitHub-Pages: https://texttechnologylab.github.io/RT-SEG/ and in Appendix A.

[3] Engine specifications and important hyperparameters can be found in Table 2 and on GitHub.

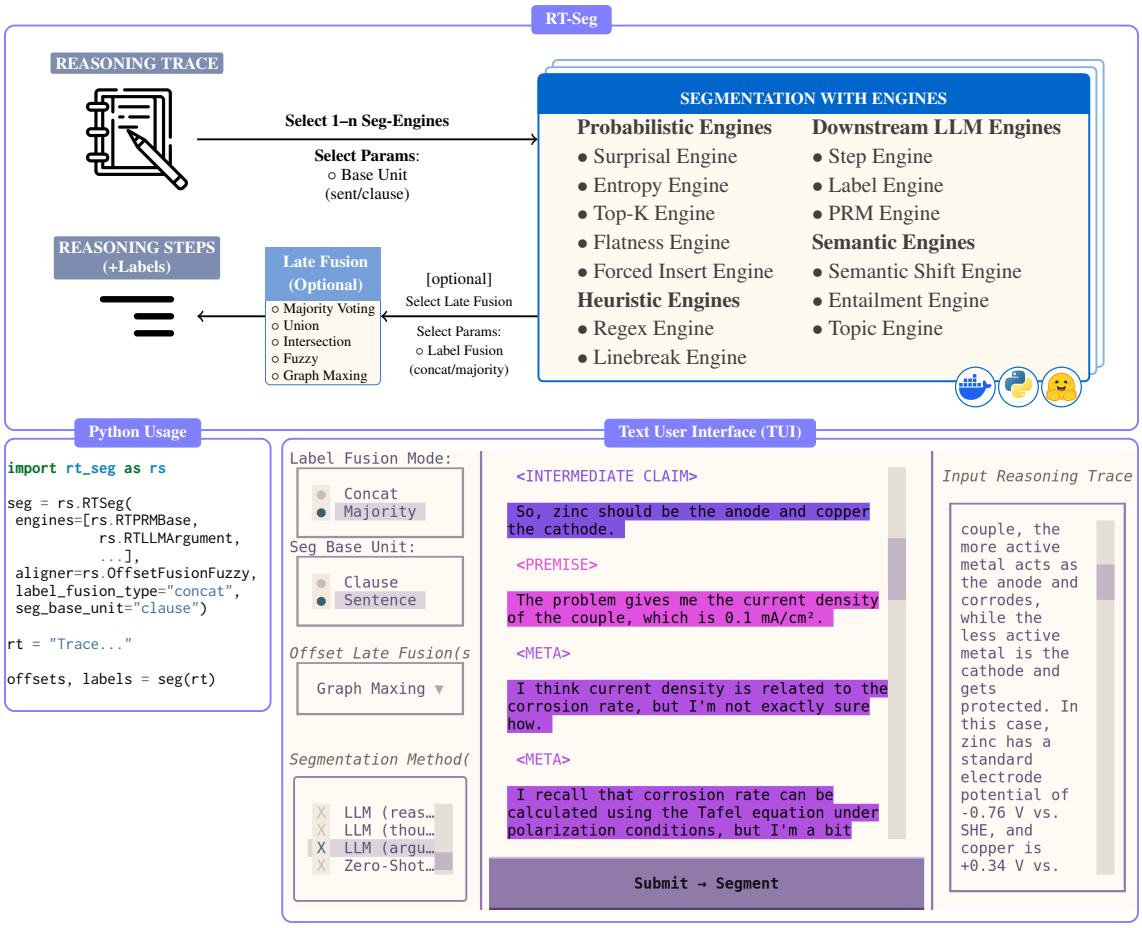

Figure 1: Overview of RT-Seg. The system architecture is shown above, with terminal and Python interfaces below.

2.2.2 Probabilistic Engines

We implement modular segmentation engines that detect reasoning-step boundaries using model-internal uncertainty signals, such as token-level surprisal and entropy (Jurafsky and Martin, 2026). All engines operate via a shared forced-decoding interface, extracting token-level log-probabilities under a segmentation prompt without modifying the trace. Each engine computes a scalar token-level signal s_t and identifies boundaries via change-point detection, using adaptive sliding-window thresholding rather than a fixed global threshold. For each candidate boundary position t, we compute a standardized score

$$z_t = \frac{s_t - \mu_w}{\sigma_w + \epsilon},$$

where μ_w and σ_w are the mean and standard deviation of s_t within a local window, and $\epsilon \in \mathbb{R}_{\geq 0}$ ensures numerical stability. A boundary is inserted when z_t exceeds a chosen quantile-based threshold. This normalization makes segmentation robust to trace length and model-specific calibration effects.

Surprisal-Based Engine. The surprisal engine uses token-level negative log-probability:

$$s_t = -\log p(x_t \mid x_{<t}),$$

where x_t is the tth token. Intuitively, large increases in surprisal indicate discontinuities in the model's predictive flow (Futrell et al., 2019), which may correspond to transitions between reasoning steps.

Entropy-Based Engine. This engine computes next-token predictive entropy:

$$s_t = -\sum_{v \in V} p(v \mid x_{<t}) \log p(v \mid x_{<t}).$$

Peaks in entropy reflect heightened uncertainty over continuation (Kadavath et al., 2022), which can signal epistemic shifts such as revisions or new inferential directions.

Top-k Engine. The Top-k engine, $k \in \mathbb{N}_{>0}$, tracks changes in the model's high-probability token set (Holtzman et al., 2020). Let T_t denote the

set of top-k tokens at position t. We define:

$$s_t = 1 - \frac{|T_t \cap T_{t-1}|}{k}, s_t \in [0, 1].$$

Large changes in the top-k set indicate a redistribution of probability mass, suggesting a shift in reasoning dynamics.

Flatness Break Engine. The flatness break engine measures abrupt changes in the local variance of log-probabilities:

$$s_t = \left|\mathrm{Var}_w(\log p(x))_t - \mathrm{Var}_w(\log p(x))_{t-1}\right|.$$

Sudden increases in variance correspond to breaks in locally smooth predictive behavior.

Forced Insert Engine In this approach, we combine forced decoding with special token insertion: the model is prompted with the reasoning trace and a system instruction specifying the segmentation task, while a separator token (e.g., <|seg|>) is added to the tokenizer. At each token position, the model evaluates the log-probability of the true next token against the log-probabilities of candidate separator tokens. Segmentation boundaries are placed whenever the normalized gap between these probabilities exceeds an adaptive threshold, optionally weighted by punctuation cues.

2.2.3 Semantic Engines

Reasoning traces often contain semantically coherent phases that recur across different parts of the trace. To capture such patterns, we implement engines based on topic modeling (Riedl and Biemann, 2012) (cf. Reynar 1999 for an early approach to topic-related segmentation), semantic similarity (Hearst, 1997), and textual entailment (Dagan et al., 2013).

Topic Engine The Topic Engine clusters clause- or sentence-level snippets using HDBSCAN (McInnes et al., 2017) over embedding representations. Each cluster is optionally labeled by an LLM, and consecutive snippets assigned to the same cluster are merged into a single segment.

Entailment Engines We employ two distinct entailment engines. The first approach leverages a transformer-based model (Lewis et al., 2020) trained for zero-shot classification of text relative to predefined labels (Yin et al., 2019). The second strategy relies on pre-trained natural language inference (NLI) models, which classify pairs of sequences as entailing, contradicting, or neutral (Williams et al., 2018). Our hypothesis is that sequences exhibiting entailment or neutrality are likely to form coherent text chunks, enabling their aggregation into meaningful segments.

Semantic Shift Engine We implement a semantic shift engine that segments traces based on abrupt drops in semantic similarity between consecutive snippets (cf. Kozima 1993). When similarity falls below a given threshold, a new segment is initiated, capturing transitions between reasoning phases.

2.2.4 Downstream LLM Engines

For all downstream LLM-based segmentation engines, we first split the reasoning trace into base units (clauses or sentences) and then group these into chunks. Chunking mitigates context length limitations of LLMs, preventing single traces from overwhelming the model. Each chunk is assigned dynamically: if a reasoning step begins in one chunk and continues into the next, the model can extend the open segment seamlessly into subsequent chunks.

Label Engines We implemented multiple logic engines based on the Toulmin argumentation framework (Toulmin, 2003) and tailored to the logical steps commonly observed in reasoning traces, as described in Bogdan et al. (2025) and Lee et al. (2025). Each engine operates as a downstream task for LLMs: models are prompted to classify base units into a set of predefined labels corresponding to the engine's logical categories. Adjacent base units sharing the same label are then merged into a single reasoning step, producing a segmented trace aligned with the underlying logical structure.

Step Engines We implemented LLM-based segmentation engines that operate directly on chunks of base units: guided by a prompt describing the segmentation task, the model predicts either segment offsets or groups base units into reasoning steps. This allows the model to infer segment boundaries without relying on predefined labels.

PRM Engine We leveraged a process-reward model (PRM) (cf. Wang et al. 2024) to detect reasoning step boundaries. Given a problem and its generated reasoning trace, the model predicts step rewards at the text base-unit level, reflecting its training on reasoning traces. Segmentation boundaries are then identified by detecting significant

changes in these reward scores over sliding windows, with adaptive thresholding applied to account for local variability (see subsubsection 2.2.2).

2.3 Late Fusion

To combine outputs from multiple segmentation engines, we provide a suite of late-fusion aligners. Each aligner is implemented as a Python class inheriting from a shared base class and must implement the `fuse(segs:list[list[tuple]])` method: Given a target reasoning trace, the aligner takes a set of candidate segmentations from one or more engines and produces a single, unified segmentation. **Voting** retains boundaries supported by a majority of engines, **Union** preserves all proposed splits while the conservative **Intersection** retains only boundaries agreed upon by all engines, **Fuzzy** merges boundaries within a small character distance to account for minor offset discrepancies, and **Graph-Based** treats each proposed segment as a weighted edge and uses dynamic programming to find the path maximizing total support, yielding a globally consistent segmentation that balances local agreement and overall coherence.

3 Segmentation Evaluation

We evaluate RT-Seg for segmentation boundary detection by comparing automatic segmentations against human-annotated reasoning traces under two annotation schemes: *ReasoningFlow* (Lee et al., 2025) and *Thought Anchor* (Bogdan et al., 2025). A complementary analysis of segment-level label distributions appears in subsection 3.4; a qualitative case study illustrating RT-Seg's advantages appears in subsection 3.3.

3.1 Experimental Setup

We conduct experiments on 100 reasoning traces generated by QwQ-32B-Preview (Yang et al., 2024) from Sky-T1 (NovaSky, 2025), following Lee et al. (2025). For ReasoningFlow, we use the human-segmented, annotated traces from Lee et al. (2025), labeled by a single annotator following their scheme. In addition, we collect new annotations for all traces under the Thought Anchor (Bogdan et al., 2025) schema. Two independent annotators freely segmented all 100 traces (i.e., without sentence-level constraints), which yielded 2,666 annotated reasoning steps for annotator 1 and 2,810 annotated reasoning steps for annotator 2. Segmentation quality is evaluated using boundary similarity (B_{sim}; Fournier 2013), which measures alignment between predicted and reference boundaries while allowing near-miss matches. We use a three-character window. For ReasoningFlow, which provides a single human reference segmentation, we compute pairwise B_{sim} between RT-Seg output and the human annotation. For Thought Anchor, which has two human annotations, we compute triadic B_{sim} by averaging over all pairwise combinations of segmentations (h1, h2, RT-Seg):

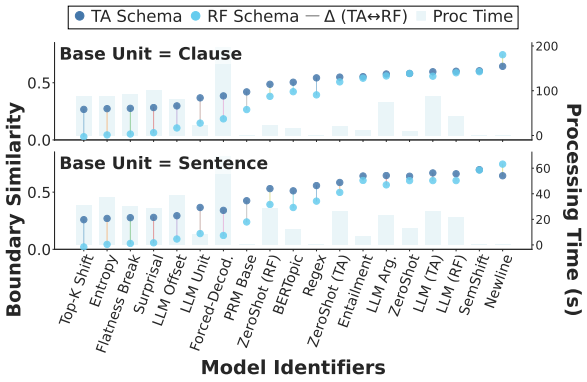

Figure 2: RT-Seg engines evaluated against the human Thought Anchor and ReasoningFlow reference segmentations: average boundary similarity and runtime for sentence- and clause-level base units.

$$B_{\text{triadic}} = \frac{1}{\binom{3}{2}} \sum_{(i,j)\in\text{pairs}} B_{\text{sim}}(S_i, S_j) \in [0,1].$$

Implementation details, including the exact RT-Seg configurations, engine hyperparameters, model choices, prompt keys, and instructions for reproducing all experiments, are provided in Appendix A.

3.2 Quantitative Results

Human reference agreement. We first report boundary similarity between human segmentations to contextualize RT-Seg performance. For the Thought Anchor schema, the two independent human annotations achieve $B_{sim}= 0.73$. When incorporating the ReasoningFlow human segmentation into a triadic evaluation, we obtain $B_{\text{triadic}}(h_{ta1}, h_{ta2}, h_{rf1})= 0.72$. This suggests that cross-schema boundary differences are comparable in magnitude to within-schema variation.

Single-engine performance. We evaluate all engines under both annotation schemes and base-unit variants (Figure 2), reporting boundary similarity averaged over all 100 traces. Probabilistic engines perform poorly (avg. $B_{sim}= 0.18$) while exhibiting high processing times (76s per

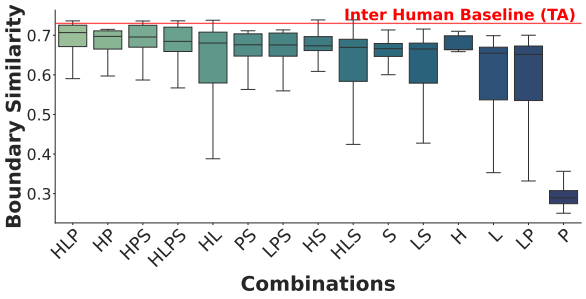

Figure 3: This figure shows all 5M configurations evaluated during the evolutionary search over the full space of RT-Seg engine combinations and offset fusion strategies. Configurations are grouped by the types of engines they include. We use the following abbreviations: S – Semantic; P – Probabilistic; L – Downstream LLM; H – Heuristic.

trace). In contrast, semantic and LLM-based engines achieve substantially stronger performance (B_{sim}= 0.54 and 0.43) with moderate to high runtime (14s and 35s, respectively). Heuristic engines perform best overall (B_{sim}= 0.59) and are also the most efficient (0.5s per trace). Stronger engines align comparably under both schemes, whereas weaker ones align slightly better with Thought Anchor. Sentence-level segmentation scores slightly higher than clause-level (B_{sim}= 0.42 vs 0.41) and is much faster (21s vs. 51s; $\approx 2.4\times$).

Ensemble search. We next explore engine combinations via evolutionary search over engine-base-unit configurations and offset-fusion strategies. Our search space consists of non-empty subsets of 38 engine-base-unit variants and 5 late-fusion methods, yielding $(2^{38} - 1) \times 5 \approx 2^{38} \times 5$ possible configurations. The 38 variants arise from 19 engines, each of which can operate independently at either sentence or clause level. Given this large space, we ran a generational genetic algorithm for approximately 800 generations, yielding roughly five million unique evaluated configurations after deduplication. The search uses a population size of 4,096, 16,384 offspring per generation, uniform crossover, bit-flip mutation with rate $1/38$, fusion-method mutation, 6.25% elitism, and tournament selection with $k = 7$. Fitness is computed as boundary similarity to the human annotations under the Thought Anchor scheme. We use this search as an exploratory analysis of the configuration space rather than as a held-out model-selection procedure. Accordingly, the best searched configurations should be interpreted as high-scoring configurations within this annotated sample, not as unbiased estimates of generalization performance.

Engine-family effects. Figure 3 shows the distribution of triadic boundary similarity scores for engine combinations relative to the inter-human boundary similarity reference score for the Thought Anchor scheme. Although probabilistic engines perform poorly in isolation, they are present in all four of the highest-scoring searched ensembles, suggesting that they provide complementary boundary signals when combined with stronger engine types. These ensembles cluster near the observed inter-human reference score, with the highest-scoring searched configurations slightly exceeding it at $B_{triadic} \approx 0.74$. Heuristic engines also remain important in the searched ensemble setting: they appear in all searched configurations that exceed the inter-human reference score, suggesting that they provide strong base segmentations that can be refined by other engine types. Overall, within the explored configuration space, heterogeneous engine combinations tend to outperform configurations restricted to a single engine type.

Fusion strategies. Finally, we analyze the effect of offset fusion strategies by grouping configurations by fusion type and estimate kernel density distributions over their achieved boundary similarity scores (Figure 4). Intersection fusion performs consistently poorly, with a narrow density concentrated around $B_{triadic} \approx 0.21$, indicating that this strategy is overly restrictive. Majority voting exhibits the widest performance spread, suggesting high sensitivity to the specific engine combination. Union and fuzzy fusion yield higher-scoring searched configurations overall, with densities concentrated in the 0.65-0.75 triadic boundary similarity range. Among these, union fusion shows a pronounced peak near boundary similarity of 0.75, indicating that its strongest searched configurations achieve high agreement within this annotated sample. Graph-based fusion is the most stable strategy in this search, with a narrow density centered near boundary similarity of 0.73.

3.3 Case Study

We present three qualitative examples from the Sky-T1 dataset illustrating cases where RT-Seg provides more informative segmentations than simple sentence-based splitting. (1) Sentence-level segmentation can be overly fine-grained, fragmenting a single reasoning step (Figure 5). (2) Segmentation is too coarse-grained, missing transi-

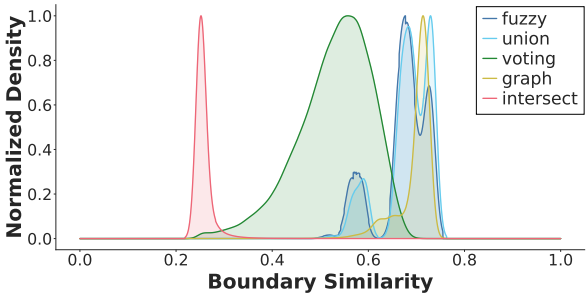

Figure 4: RT-Seg configurations grouped by fusion type. The plot shows kernel density estimates of their achieved boundary similarity scores.

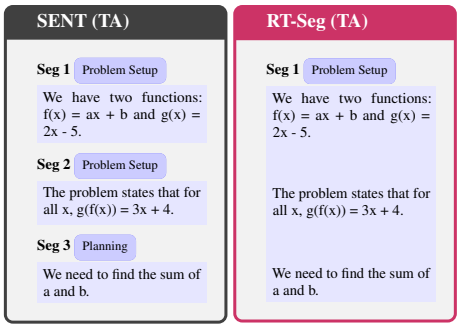

Figure 5: (Case 1) Sentence-level segmentation is too fine-grained. Comparison of reasoning trace segmentation using simple sentence splitting (left) and RT-Seg (right), both under the Thought Anchor (TA) schema.

tions across sentence boundaries (Lee et al., 2025) (Figure 6). (3) RT-Seg enables the integration of Thought Anchor and ReasoningFlow labeling schemes within a unified framework (Figure 7).

3.4 Structural Label Analysis

We analyze how structural labels are distributed over normalized trace positions for the Thought Anchor and ReasoningFlow annotation schemas (Figure 8 and Figure 9). For Thought Anchor, we compare our two independent human reference annotations (Human 1 and Human 2) with annotations produced by RT-Seg, using a mixed configuration that combines an LLM-based label engine with a heuristic engine under the Thought Anchor guidelines. Overall, the human and model annotations show similar positional trends for most labels. The main exception is Fact Retrieval, for which RT-Seg produces substantially higher frequencies than the human annotators while preserving similar temporal trends. For ReasoningFlow, we compare the human annotation from Lee et al. (2025) against the same RT-Seg setup, now applied under the ReasoningFlow annotation guidelines. Here, frequency differences are more pronounced. The model over-

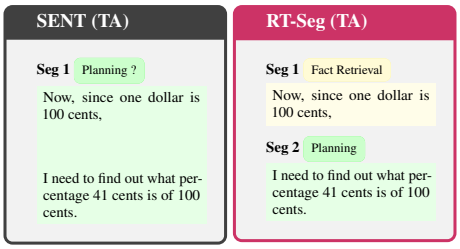

Figure 6: (Case 2) Sentence-level segmentation is too coarse-grained. Comparison of reasoning trace segmentation using simple sentence splitting (left) and RT-Seg (right), both under the Thought Anchor (TA) schema.

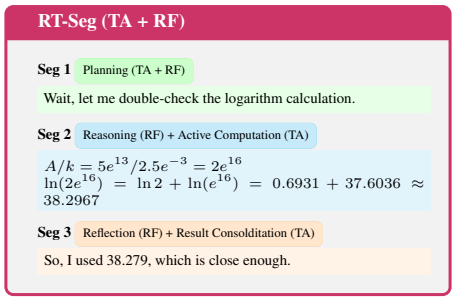

Figure 7: (Case 3) Reasoning trace segmentation with RT-Seg, combining concepts from the Thought Anchor and the ReasoningFlow schemas.

produces Assumption labels and places Conclusion labels more broadly throughout the sequence, whereas the human annotation concentrates Conclusion labels more strongly toward the end. One possible explanation is that RT-Seg treats intermediate results as conclusions, analogous to Result Consolidation in the Thought Anchor schema. Human annotations also show higher frequencies for Reasoning and Planning. At the same time, Fact, Restatement, and Reflection exhibit comparable positional patterns across human and model annotations.

4 Downstream Evaluation

We evaluate the downstream utility of the segment boundaries produced by RT-Seg on two tasks: classifying whether a reasoning prefix ends at the onset of the first error and determining whether two segments are adjacent in the original reasoning trace.

4.1 Shared Experimental Setup

Dataset. We combine the Phase 2 training and test partitions of PRM800K (Lightman et al., 2024) and retain traces whose annotation terminates with `found_error`. For each segmentation configuration, we select the same 5,000 reasoning traces and resegment each trace with RT-Seg. The first new

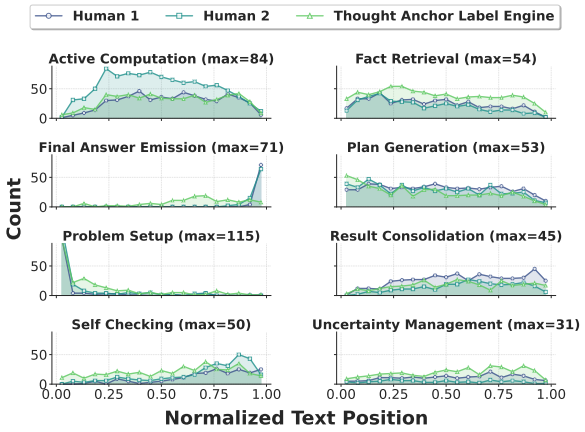

Figure 8: Temporal distribution of Thought Anchor step types across normalized trace positions. Lines show two human annotations and one LLM-based RT-Seg annotation.

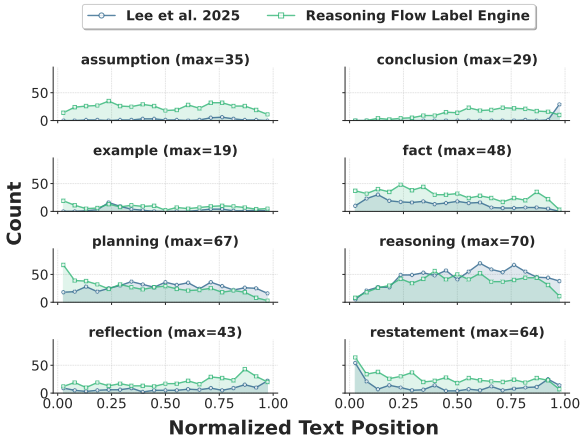

Figure 9: Temporal distribution of ReasoningFlow step types across normalized trace positions. Lines show the human annotation from Lee et al. (2025) and one LLM-based RT-Seg annotation.

segment overlapping the originally annotated error span is assigned the error-onset label; preceding segments are marked as correct, and subsequent segments remain unannotated. We compare plain sentence-level and clause-level segmentation baselines against an experimental RT-Seg configuration that applies clause-level segmentation using `Newline`, `Regex`, and `ZeroShot(TA)` components with graph-based fusion. The selected components were identified in section 3 as both high-performing and computationally efficient.

Model and training. For every task and segmentation method, we fine-tune a separate `ModernBERT-base` binary classifier (Warner et al., 2025). All models are trained for six epochs with a learning rate of 2×10^{-5}, weight decay of 0.01, a warm-up ratio of 0.1, dynamically padded batches, balanced class weights, BF16 precision, FlashAttention-2, and gradient checkpointing.

Evaluation protocol. We use five-fold stratified group cross-validation, where all samples derived from the same reasoning trace belong to the same fold. This prevents samples from the same trace from occurring in both training and test data. Within each outer training fold, approximately 20% of the data are held out as a group-disjoint calibration set. This set is used both to select the best epoch and to choose the decision threshold that maximizes macro F_1 score. The selected model and threshold are subsequently evaluated on the untouched outer test fold. We report class-specific and macro-averaged F_1 scores as the mean and standard deviation across the five outer folds.

4.2 Error-Onset Classification

The objective is to determine whether the final segment of a reasoning prefix contains the onset of the first annotated error. An input contains every segment up to and including the target segment, separated by a special `[STEP]` token. Each trace contributes one positive prefix ending at its first error. Correct prefixes preceding the error are sampled globally to construct a balanced dataset. Consequently, every configuration contains 5,000 first-error and 5,000 correct samples.

4.3 Reasoning-Step Adjacency Classification

We formulate reasoning-step ordering as a binary adjacency-classification task. Given two segments from the same trace, the classifier predicts whether the second segment immediately follows the first. Ordered adjacent pairs (s_i, s_{i+1}) are positive examples. Negative examples are sampled from ordered, non-consecutive pairs in the same trace and may therefore either skip intermediate steps or reverse their order. We sample one negative pair per positive pair, producing balanced classes. The pairs are passed to ModernBERT using its native sequence-pair encoding. Traces containing fewer than two segments are omitted. We sample 50,000 positive and 50,000 negative pairs for each segmentation condition.

4.4 Results

As shown in Table 1, the evaluated RT-Seg configuration outperforms sentence-based and clause-based segmentation on both tasks. It improves

	Error-Onset			Reasoning-Step Adjacency		
Segmentation	Correct	Error onset	Macro	Not following	Following	Macro
Sentence	67.51 ± 1.19	70.20 ± 0.49	68.85 ± 0.43	58.80 ± 1.71	58.12 ± 1.62	58.46 ± 0.28
Clause	68.72 ± 1.90	68.31 ± 2.90	68.51 ± 0.72	54.27 ± 0.59	56.25 ± 0.77	55.26 ± 0.42
RT-Seg	$\mathbf{70.28 \pm 1.11}$	$\mathbf{70.23 \pm 0.63}$	$\mathbf{70.25 \pm 0.51}$	$\mathbf{60.00 \pm 0.91}$	$\mathbf{67.43 \pm 0.44}$	$\mathbf{63.72 \pm 0.53}$

Table 1: Downstream results in percent, reported as mean $\pm$ standard deviation over five folds.

mean macro F_1 by 1.40 points on error-onset classification (70.25 vs. 68.85) and by 5.26 points on reasoning-step adjacency classification (63.72 vs. 58.46) over the best baseline.

5 Related Work

Reasoning Traces in LLM Analysis. Reasoning traces are widely used as artifacts for analyzing LLM behavior, including model performance, failure modes, consistency, and alignment (Lee and Hockenmaier, 2025; Abdollahi et al., 2026; Stechly et al., 2024). These analyses often inspect traces as generated, without formally segmenting them into coherent reasoning steps. While valuable for understanding model behavior, this approach treats the trace as a whole (or only coarsely segmented), leaving reasoning-step structure implicit.

Structure and Compositionality of Reasoning. Most prior work operationalizes reasoning steps using surface-level textual units such as sentences, line breaks, or prompt-defined separators (Bogdan et al., 2025; Nguyen et al., 2024; Golovneva et al., 2023). At the same time, reasoning traces admit multiple valid segmentations that differ in granularity and conceptual framing (Xiong et al., 2025; Beck et al., 2020; Lee et al., 2025). Different works emphasize distinct perspectives: some interpret steps through epistemic or cognitive functions (Bogdan et al., 2025), others focus on semantic roles (Lee et al., 2025), and still others segment traces into linguistically motivated *subthoughts* (Hammoud et al., 2025; Jiang et al., 2025). In practice, most analyses commit to a single granularity and rarely compare alternatives, which can affect step-level evaluation, model comparison, and alignment with human judgments.

Text Segmentation and Discourse Structure. Reasoning trace segmentation is related to discourse parsing, argument mining, and procedural step extraction (Lawrence and Reed, 2019; Stab and Gurevych, 2017; Joty et al., 2015). While these areas address structural decomposition of text, they primarily focus on linguistic or argumentative organization rather than flexible, model-dependent reasoning steps produced by LLMs.

6 Conclusion

We presented RT-Seg, a configurable framework for reasoning trace segmentation that integrates heterogeneous engines (19), fusion strategies (5), and annotation schemas (3) within a single system. In contrast to prior approaches that commit to a single segmentation strategy, RT-Seg enables structured comparison and composition of alternative reasoning-step definitions. Rather than prescribing a single optimal segmentation strategy, RT-Seg is intended as an infrastructure for systematically exploring diverse reasoning-step formulations. A key aspect of RT-Seg is its lightweight architecture as a pure-Python program, which facilitates the integration of additional segmentation engines and strategies through inheritance. RT-Seg therefore serves as a toolkit for comparative reasoning-trace analysis. Our experiments show that, within the explored configuration space, ensemble configurations tend to achieve higher boundary similarity than single-engine approaches and can approach human reference segmentations under the evaluated annotation schemes. Moreover, the tested RT-Seg configuration improves over sentence-based segmentation on both error-onset classification and reasoning-step adjacency classification, indicating that its segments provide more useful units for downstream reasoning analysis. The highest-scoring searched ensembles combine heterogeneous engine types, suggesting that different segmentation strategies may capture complementary aspects of reasoning structure. These findings support the central design of RT-Seg: a modular ensemble-based framework that makes such combinations easy to construct, evaluate, and compare. By releasing RT-Seg as an open-source Python library with a Dockerized TUI, we provide practical infrastructure for systematic and reproducible reasoning-trace research.

Limitations

Our main intrinsic metric, boundary similarity, measures structural agreement between segment boundaries but does not directly evaluate semantic coherence or label correctness. We therefore complement it with two downstream tasks; however, these tasks cover only mathematical reasoning traces and should not be interpreted as establishing utility for all reasoning-analysis settings. Consequently, high boundary similarity should not be interpreted as sufficient evidence that a segmentation is optimal for every analysis. Similarly, configurations that approach or slightly exceed observed inter-human agreement should be interpreted as strongly aligned under the chosen metric, not as inherently superior to human annotations. Finally, RT-Seg exposes a large configuration space of engines, base units, and fusion strategies. Although this flexibility is central to the toolkit, selecting strong configurations may require task-specific tuning. In addition, components that rely on LLMs or model-internal uncertainty signals may be sensitive to model choice, prompts, calibration, and context-length constraints.

Acknowledgements

This work was supported by the German Research Foundation (DFG) as part of Project INF within CRC 1629 "Negation in Language and Beyond" (NegLaB - https://www.neglab.de/).

References

Mohammad Abdollahi, Khandaker Rifah Tasnia, Soumit Kanti Saha, Jinqiu Yang, Song Wang, and Hadi Hemmati. 2026. Demystifying errors in LLM reasoning traces: An empirical study of code execution simulation. *ACM Transactions on Software Engineering and Methodology*. Just Accepted.

Christin Beck, Hannah Booth, Mennatallah El-Assady, and Miriam Butt. 2020. Representation problems in linguistic annotations: Ambiguity, variation, uncertainty, error and bias. In *Proceedings of the 14th Linguistic Annotation Workshop*, pages 60–73, Barcelona, Spain. Association for Computational Linguistics.

Paul C. Bogdan, Uzay Macar, Neel Nanda, and Arthur Conmy. 2025. Thought anchors: Which LLM reasoning steps matter? In *Mechanistic Interpretability Workshop at NeurIPS 2025*.

Ido Dagan, Dan Roth, Mark Sammons, and Fabio Massimo Zanzotto. 2013. *Recognizing Textual Entailment: Models and Applications*. Synthesis Lectures on Human Language Technologies. Springer International Publishing.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2024. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11733–11763. PMLR.

Chris Fournier. 2013. Evaluating text segmentation using boundary edit distance. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1702–1712, Sofia, Bulgaria. Association for Computational Linguistics.

Richard Futrell, Ethan Wilcox, Takashi Morita, Peng Qian, Miguel Ballesteros, and Roger Levy. 2019. Neural language models as psycholinguistic subjects: Representations of syntactic state. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 32–42, Minneapolis, Minnesota. Association for Computational Linguistics.

Olga Golovneva, Moya Chen, Spencer Poff, Martin Corredor, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. 2023. ROSCOE: A suite of metrics for scoring step-by-step reasoning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638.

Hasan Abed Al Kader Hammoud, Hani Itani, and Bernard Ghanem. 2025. Beyond the last answer: Your reasoning trace uncovers more than you think. *Preprint*, arXiv:2504.20708.

Marti A. Hearst. 1997. TextTiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):33–64.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Gangwei Jiang, Yahui Liu, Zhaoyi Li, Wei Bi, Fuzheng Zhang, Linqi Song, Ying Wei, and Defu Lian. 2025. What makes a good reasoning chain? Uncovering structural patterns in long chain-of-thought reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages

6490–6514, Suzhou, China. Association for Computational Linguistics.

Shafiq Joty, Giuseppe Carenini, and Raymond T. Ng. 2015. CODRA: A novel discriminative framework for rhetorical analysis. *Computational Linguistics*, 41(3):385–435.

Daniel Jurafsky and James H. Martin. 2026. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models*, 3rd edition. Online manuscript released January 6, 2026.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, and 17 others. 2022. Language models (mostly) know what they know. *Preprint*, arXiv:2207.05221.

Tomek Korbak, Mikita Balesni, Elizabeth Barnes, Yoshua Bengio, Joe Benton, Joseph Bloom, Mark Chen, Alan Cooney, Allan Dafoe, Anca D. Dragan, Scott Emmons, Owain Evans, David Farhi, Ryan Greenblatt, Dan Hendrycks, Marius Hobbhahn, Evan Hubinger, Geoffrey Irving, Erik Jenner, and 22 others. 2025. Chain of thought monitorability: A new and fragile opportunity for AI safety. *CoRR*, abs/2507.11473.

Hideki Kozima. 1993. Text segmentation based on similarity between words. In *31st Annual Meeting of the Association for Computational Linguistics*, pages 286–288, Columbus, Ohio, USA. Association for Computational Linguistics.

John Lawrence and Chris Reed. 2019. Argument mining: A survey. *Computational Linguistics*, 45(4):765–818.

Jinu Lee and Julia Hockenmaier. 2025. Evaluating step-by-step reasoning traces: A survey. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 1789–1814, Suzhou, China. Association for Computational Linguistics.

Jinu Lee, Sagnik Mukherjee, Dilek Hakkani-Tur, and Julia Hockenmaier. 2025. ReasoningFlow: Semantic structure of complex reasoning traces. *Preprint*, arXiv:2506.02532.

Yiming Lei, Jiawei Xu, Chia Xin Liang, Ziqian Bi, Xiaoming Li, Danyang Zhang, Junhao Song, and Zhenyu Yu. 2025. Reasoning in large language models: From chain-of-thought to massively decomposed agentic processes. *Preprints*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594. Curran Associates, Inc.

Leland McInnes, John Healy, and Steve Astels. 2017. HDBSCAN: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205.

Minh-Vuong Nguyen, Linhao Luo, Fatemeh Shiri, Dinh Phung, Yuan-Fang Li, Thuy-Trang Vu, and Gholamreza Haffari. 2024. Direct evaluation of chain-of-thought in multi-hop reasoning with knowledge graphs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2862–2883, Bangkok, Thailand. Association for Computational Linguistics.

NovaSky. 2025. Sky-T1: Train your own o1-preview model within $450. Accessed: 2025-01-09.

Jeffrey C. Reynar. 1999. Statistical models for topic segmentation. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 357–364, College Park, Maryland, USA. Association for Computational Linguistics.

Martin Riedl and Chris Biemann. 2012. TopicTiling: A text segmentation algorithm based on LDA. In *Proceedings of ACL 2012 Student Research Workshop*, pages 37–42, Jeju Island, Korea. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551. Curran Associates, Inc.

Zhao Song, Song Yue, and Jiahao Zhang. 2025. Thinking isn't an illusion: Overcoming the limitations of reasoning models via tool augmentations. *SuperIntelligence - Robotics - Safety & Alignment*, 2(6).

Christian Stab and Iryna Gurevych. 2017. Parsing argumentation structures in persuasive essays. *Computational Linguistics*, 43(3):619–659.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. 2024. Chain of thoughtlessness? An analysis of CoT in planning. In *Advances in Neural Information Processing Systems*, volume 37, pages 29106–29141. Curran Associates, Inc.

Stephen E. Toulmin. 2003. *The Uses of Argument*, 2nd edition. Cambridge University Press.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Advances in Neural Information Processing Systems*, volume 36, pages 74952–74965. Curran Associates, Inc.

Jonathan Uesato, Nate Kushman, Ramana Kumar, H. Francis Song, Noah Y. Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process- and outcome-based feedback. *CoRR*, abs/2211.14275.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-Shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, Bangkok, Thailand. Association for Computational Linguistics.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Griffin Thomas Adams, Jeremy Howard, and Iacopo Poli. 2025. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2526–2547, Vienna, Austria. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Yeo Wei Jie, Ranjan Satapathy, Rick Goh, and Erik Cambria. 2024. How interpretable are reasoning explanations from prompting large language models? In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2148–2164, Mexico City, Mexico. Association for Computational Linguistics.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Zhen Xiong, Yujun Cai, Zhecheng Li, and Yiwei Wang. 2025. Mapping the minds of LLMs: A graph-based analysis of reasoning LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17751–17763, Suzhou, China. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 43 others. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Wenpeng Yin, Jamaal Hay, and Dan Roth. 2019. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3914–3923, Hong Kong, China. Association for Computational Linguistics.

Sheldon Yu, Yuxin Xiong, Junda Wu, Xintong Li, Tong Yu, Xiang Chen, Ritwik Sinha, Jingbo Shang, and Julian McAuley. 2025. Explainable chain-of-thought reasoning: An empirical analysis on state-aware reasoning dynamics. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 16660–16667, Suzhou, China. Association for Computational Linguistics.

Zi'ou Zheng, Christopher Malon, Martin Renqiang Min, and Xiaodan Zhu. 2024. Exploring the role of reasoning structures for constructing proofs in multi-step natural language reasoning with large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15299–15312, Miami, Florida, USA. Association for Computational Linguistics.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The*

Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net.

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. 2024. Self-Discover: Large language models self-compose reasoning structures. In *Advances in Neural Information Processing Systems*, volume 37, pages 126032–126058. Curran Associates, Inc.

A Reproducibility Details

The code used in this paper is available in the project repository: https://github.com/texttechnologylab/RT-SEG and https://doi.org/10.5281/zenodo.21416283. The repository contains the implementation of RT-Seg, the annotated dataset, scripts for reproducing the experiments, configuration files, Docker setup, and instructions for running the text user interface (TUI). The annotated dataset can be downloaded from the repository and hosted via SurrealDB for reproducing the database-backed experiments. The Dockerized TUI supports exploratory experimentation with custom reasoning traces and engine configurations. In addition, RT-Seg can be installed either directly from the GitHub repository or via the published PyPI package: https://pypi.org/project/rt-seg/.

A.1 Experimental Setup

All experiments in this paper use the default engine configurations provided by RT-Seg. These defaults are implemented in the corresponding engine classes, in particular in each engine's `_segment(...)` method, and in the global RT-Seg configuration under `src/seg_factory`. We summarize the relevant defaults, models, and prompt keys in Table 2 and display the default prompts in Appendix D.

A.2 SurrealDB Dataset and Experiment Commands

The repository contains a SurrealDB export at `surrealdb_dataset/RT_RF_extended.surql`. A local reproduction can use the following setup:

```
python3.12 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt

docker run --rm -it \
  -p 8000:8000 \
  -v "$(pwd)/data:/data" \
  surrealdb/surrealdb:latest \
  start --user root --pass root file:/data/surreal.db

surreal import \
  --endpoint ws://127.0.0.1:8000/rpc \
  --username root \
  --password root \
  --namespace NR \
  --database RT_RF_extended \
  surrealdb_dataset/RT_RF_extended.surql
```

The runtime database connection is read from `data/sdb_login.json`. For a local instance, the file has the following shape:

```
{
  "user": "root",
  "pwd": "root",
  "ns": "NR",
  "db": "RT_RF_extended",
  "url": "ws://127.0.0.1:8000/rpc"
}
```

A segmentation run over the database can be launched from Python:

```
from rt_seg import (
    RTSeg, RTRuleRegex, RTNewLine,
    ↪  RTEmbeddingBasedSemanticShift,
    OffsetFusionGraph,
)

segmentor = RTSeg(
    engines=[RTEmbeddingBasedSemanticShift, RTRuleRegex,
    ↪  RTNewLine],
    aligner=OffsetFusionGraph,
    seg_base_unit="clause",
    label_fusion_type="majority",
)
segmentor.sdb_segment_ds(clear=True, db="RT_RF_extended")
```

The evaluation and search entry points used by the paper are:

```
python src/eval_main.py   # scoring helpers and plot
↪  generation
python src/evo.py         # evolutionary ensemble search
python src/test.py        # test runs for all engines
```

B Python Package Usage

The python package can be found and installed from PyPI or Github (see Appendix A). The notebook `notebook/quick_start.ipynb` from the github repository demonstrates the public Python interface.

B.1 Basic Usage

The simplest pattern is to import the package, instantiate `RTSeg`, call it on a trace, and reconstruct the text spans from returned offsets.

```
import rt_seg as rt

segmentor = rt.RTSeg(
    engines=[rt.RTEmbeddingBasedSemanticShift,
    ↪  rt.RTRuleRegex, rt.RTNewLine],
    aligner=rt.OffsetFusionGraph,
    seg_base_unit="clause",
    label_fusion_type="majority",
)

offsets, labels = segmentor(rtrace)
segments = [rtrace[s:e] for s, e in offsets]
```

B.2 Custom Parameters

Engine keyword arguments can be overridden at call time. For example, the zero-shot classifier can be run with a task-specific label set:

```
segmentor = rt.RTSeg(
    engines=[rt.RTZeroShotSeqClassification],
    seg_base_unit="clause",
    label_fusion_type="majority",
```

Table 2: Default models, prompts, and explicit engine keyword arguments. The keyword column is derived from the `_segment` signatures in `src/rt_segmentation`, excluding only `trace` and trailing `**kwargs`. If `RTSeg` supplies a required keyword or overrides a signature default, this is stated inline.

Engine	Explicit `_segment` keyword arguments and defaults
Heuristic engines	
`RTRuleRegex`	`seg_base_unit` (factory: `clause`; can be `sent`).
`RTNewLine`	No explicit keyword arguments.
Probabilistic engines	
`RTLLMForcedDecoderBased`	`system_prompt` (factory: `system_prompt_forceddecoder`); `model_name` (factory: `model_base`, default `Qwen/Qwen2.5-0.5B-Instruct`); `max_kv_tokens=512`; `alpha=1.0`; `beta=2`; `quantile=90.0`; `sep_tok="<|seg|>"`.
`RTLLMSurprisal`	`system_prompt` (factory: `system_prompt_surprisal`); `model_name` (factory: `model_base`, default `Qwen/Qwen2.5-0.5B-Instruct`); `max_kv_tokens=512`; `window=15`; `quantile=10`.
`RTLLMEntropy`	`system_prompt` (factory: `system_prompt_surprisal`); `model_name` (factory: `model_base`, default `Qwen/Qwen2.5-0.5B-Instruct`); `max_kv_tokens=512`; `window=30`; `quantile=10`.
`RTLLMTopKShift`	`system_prompt` (factory: `system_prompt_surprisal`); `model_name` (factory: `model_base`, default `Qwen/Qwen2.5-0.5B-Instruct`); `max_kv_tokens=512`; `top_k=20`; `window=15`; `quantile=90`.
`RTLLMFlatnessBreak`	`system_prompt` (factory: `system_prompt_surprisal`); `model_name` (factory: `model_base`, default `Qwen/Qwen2.5-0.5B-Instruct`); `max_kv_tokens=512`; `window=15`; `quantile=10`.
Semantic engines	
`RTBERTopicSegmentation`	`seg_base_unit` (factory: `clause`); `model_name` (factory: `Qwen/Qwen2.5-1.5B-Instruct`); `embedding_model_name="all-MiniLM-L6-v2"`; `system_prompt=""` (factory overrides with `system_prompt_topic_label`); `all_custom_labels=False`.
`RTEmbeddingBasedSemanticShift`	`seg_base_unit` (factory: `clause`); `model_name="all-MiniLM-L6-v2"`; `tolerance=0.15`; `min_threshold=0.4`.
`RTEntailmentBasedSegmentation`	`seg_base_unit` (factory: `clause`); `tolerance=0.15`; `min_threshold=0.25`; `model_name="MoritzLaurer/mDeBERTa-v3-base-xnli-multilingual-nli-2mil7"`.
`RTZeroShotSeqClassification`	`seg_base_unit` (factory: `clause`); `model_name` (factory: `facebook/bart-large-mnli`); `labels=["Context","Planning","Fact","Restatement","Example","Reflection","Conclusion"]` (signature default; factory overrides with `[verification,pivot,inference,framing,conclusion]`).
`RTZeroShotSeqClassificationRF`	`seg_base_unit` (factory: `clause`); `model_name` (factory: `facebook/bart-large-mnli`); `labels=[Context,Planning,Fact,Reasoning,Restatement,Assumption,Example,Reflection,Conclusion]` (fixed by subclass partial).
`RTZeroShotSeqClassificationTA`	`seg_base_unit` (factory: `clause`); `model_name` (factory: `facebook/bart-large-mnli`); `labels=[Problem Setup,Plan Generation,Fact Retrieval,Active Computation,Uncertainty Management,Result Consolidation,Self Checking,Final Answer Emission]` (fixed by subclass partial).
Downstream LLM engines	
`RTPRMBase`	`seg_base_unit` (factory: `clause`); `problem="The problem is unknown, please reason step by step."`; `chunk_size=50`; `window=4`; `quantile=60`. Factory also passes `model_name=Qwen/Qwen2.5-Math-7B-PRM800K`, but this is consumed through `**kwargs` rather than an explicit signature keyword.
`RTLLMThoughtAnchor`	`seg_base_unit` (factory: `clause`); `user_prompt` (factory: `user_prompt_thought_anchor`); `system_prompt` (factory: `system_prompt_thought_anchor`); `model_name` (factory: `Qwen/Qwen2.5-7B-Instruct`); `max_retry=30`; `context_window=2`.
`RTLLMReasoningFlow`	`seg_base_unit` (factory: `clause`); `user_prompt` (factory: `user_prompt_reasoning_flow`); `system_prompt` (factory: `system_prompt_reasoning_flow`); `model_name` (factory: `Qwen/Qwen2.5-7B-Instruct`); `max_retry=30`; `context_window=2`.
`RTLLMArgument`	`seg_base_unit` (factory: `clause`); `system_prompt` (factory: `system_prompt_argument`); `user_prompt` (factory: `user_prompt_argument`); `model_name` (factory: `Qwen/Qwen2.5-7B-Instruct`); `context_window=2`; `max_retry=30`.
`RTLLMOffsetBased`	`chunk_size` (factory: `300`); `prompt` (factory: empty string); `system_prompt` (factory: `system_prompt_offset`); `model_name` (factory: `Qwen/Qwen2.5-7B-Instruct`); `margin=200`; `max_retries_per_chunk=10`.
`RTLLMSegUnitBased`	`seg_base_unit` (factory: `clause`); `chunk_size` (factory: `100`); `prompt` (factory: empty string); `system_prompt` (factory: `system_prompt_sentbased`); `model_name` (factory: `Qwen/Qwen2.5-7B-Instruct`); `max_retry=30`.

```
)

offsets, labels = segmentor(
    rtrace,
    labels=["Problem", "Solution", "Intermediate"],
)
```

B.3 Subclassing Engines and Aligners

Custom engines only need to subclass `SegBase` and implement the static `_segment(...)` method. Custom aligners subclass `OffsetFusion` and implement `fuse`. The notebook uses the following minimal example:

```
class CustomEngine(rt.SegBase):
    @staticmethod
    def _segment(
        trace: str,
        breakpoint: int,
        **kwargs,
    ) -> tuple[list[tuple[int, int]], list[str]]:
        return [(0, breakpoint), (breakpoint, len(trace))],
        ↪  ["1", "2"]


class CustomAligner(rt.OffsetFusion):
    @staticmethod
    def fuse(
        segmentations: list[list[tuple[int, int]]],
        **kwargs,
    ) -> list[tuple[int, int]]:
        return segmentations[0]


custom_segmentor = rt.RTSeg(
    engines=[CustomEngine, rt.RTRuleRegex],
    aligner=CustomAligner,
    seg_base_unit="clause",
    label_fusion_type="majority",
)

offsets, labels = custom_segmentor(rtrace, breakpoint=100)
```

C Dockerized TUI Application

RT-Seg also provides a Dockerized text user interface (TUI) for interactive exploration of reasoning-trace segmentations. The TUI allows users to enter or load traces, select segmentation engines and fusion strategies, and inspect the resulting segmented output without writing Python code. The containerized version requires an NVIDIA GPU with CUDA support. It can be built and launched with Docker as follows:

```
docker build -f docker/Dockerfile -t mytui:gpu .
docker run -it --rm --gpus all mytui:gpu
```

Alternatively, the same image can be built and executed with Podman:

```
podman build -f docker/Dockerfile -t mytui:gpu .
podman run -it --rm --device nvidia.com/gpu=all mytui:gpu
```

D Prompt Registry

Default prompts are shipped in the runtime registry `src/rt_segmentation/prompts.json` and loaded with `load_prompt`. The keys below are the prompts used by the LLM-dependent components.

D.1 Boundary-Inference Prompts

`system_prompt_offset`

```
You are a reasoning trace segmenter. Given a language
↪  model's reasoning trace, divide it into reasoning
↪  segments by inserting boundaries only between
↪  sentences.
Rules:
- Each segment must represent one coherent reasoning step —
↪  a single epistemic function.
- Insert a boundary between two sentences only when the
↪  second sentence begins a new/different epistemic
↪  function.
- Do NOT judge correctness, logic, or factual accuracy —
↪  focus only on change of epistemic role.
- Boundaries are allowed even when topic, entities or
↪  vocabulary remain largely the same.
Main epistemic functions (reference only — do not label):
1. Assumption / Setup (premises, constraints, known facts)
2. Intermediate Inference (deriving new information)
3. Contrast / Objection / Alternative
4. Revision / Correction / Reconsideration
5. Goal / Plan / Next-step management
6. Conclusion / Final answer / Summary
Hard constraints:
- Boundaries only between sentences — never split inside a
↪  sentence.
- Segments must be contiguous and non-overlapping.
- Every character belongs to exactly one segment.
Granularity:
- Do NOT split on minor paraphrases, restatements, or
↪  rewordings.
- Do NOT split continuous arithmetic/symbolic chains unless
↪  clearly interrupted by a role change.
- Do split when the epistemic role changes, even subtly.
- When in doubt → do NOT split (prefer larger segments).
OUTPUT FORMAT:
Return only a JSON array of offset pairs, nothing else.
↪  Each pair represents [start, end] character offsets
↪  (0-based, inclusive start, exclusive end) of one
↪  segment in the original text:
[
  [0, 142],
  [142, 289],
  [289, 415],
  ...
]
```

`prompt_offset`.

```
Segment the following text into reasoning segments
↪  according to the system instructions.
TEXT TO SEGMENT (use exact character positions and return
↪  the offsets in json format):
```

`system_prompt_sentbased`.

```
You are a reasoning trace segmenter. You will receive a
↪  JSON object containing the reasoning trace already
↪  split into numbered sentences.
Input format (example):
{
  "sentences": [
    {"id": 0, "text": "First sentence here."},
    {"id": 1, "text": "Second sentence here."},
    ...
  ]
}
Your task: Group these sentences into reasoning segments.
Rules:
- Each segment must represent one coherent reasoning step —
↪  a single epistemic function.
- Start a new segment only when a sentence introduces a
↪  clearly new/different epistemic function compared to
↪  the previous one.
- Do NOT judge correctness, logic, or factual accuracy —
↪  focus only on change of epistemic role.
- A topic/ entity/ vocabulary change alone is NOT enough to
↪  start a new segment.
Main epistemic functions (reference only — do not label):
1. Assumption / Setup (premises, constraints, known facts)
2. Intermediate Inference (deriving new information)
3. Contrast / Objection / Alternative
4. Revision / Correction / Reconsideration
5. Goal / Plan / Next-step management
6. Conclusion / Final answer / Summary
Granularity:
- Do NOT split on minor paraphrases, restatements,
↪  rewordings, or elaborations of the same point.
- Do NOT split continuous arithmetic/symbolic/derivation
↪  chains unless a clear role change occurs.
- Do split when the epistemic role meaningfully changes,
↪  even subtly.
- When in doubt → do NOT split (prefer larger segments /
↪  fewer boundaries).
Hard constraints:
- Segments consist of one or more consecutive sentences (by
↪  id).
- All sentences must be used exactly once.
- Segments must be contiguous (no reordering or skipping).
OUTPUT FORMAT:
Return ONLY a valid JSON array of arrays of sentence IDs,
↪  where each inner array represents one reasoning segment
↪  (in original order):
[
  [1, 2, 3],
  [4],
  [5, 6, 7, 8],
  [9, 10],
  ...
]
Do not include any other text, explanations, or keys
↪  outside this array.
```

`system_prompt_forceddecoder`.

```
You are a reasoning trace segmenter.
Given a language model's reasoning trace, segment it into
↪  reasoning steps by INSERTING A SEGMENT TOKEN at valid
↪  boundaries.
SEGMENT TOKENS:
- Step
- ' Step' (Step with a leading space)
A segment boundary is represented ONLY by emitting one of
↪  the segment tokens above.
Do NOT insert boundaries in any other way.
Rules:
- Each segment must represent one coherent reasoning step —
↪  a single epistemic function.
- Insert a segment token ONLY between two sentences, NEVER
↪  inside a sentence.
- Insert a segment token ONLY when the next sentence begins
↪  a new or different epistemic function.
- Do NOT judge correctness, logic, or factual accuracy —
↪  focus only on change of epistemic role.
- Boundaries are allowed even when topic, entities, or
↪  vocabulary remain largely the same.
Main epistemic functions (reference only — do NOT label):
1. Assumption / Setup (premises, constraints, known facts)
2. Intermediate Inference (deriving new information)
3. Contrast / Objection / Alternative
4. Revision / Correction / Reconsideration
5. Goal / Plan / Next-step management
6. Conclusion / Final answer / Summary
Hard constraints:
- Segment tokens may ONLY appear between complete sentences.
- NEVER insert a segment token mid-sentence.
- Segments must be contiguous and non-overlapping.
- Every character of the original text must belong to
↪  exactly one segment.
- If the text already naturally emits a segment token, do
↪  NOT emit an additional one.
Granularity:
- Do NOT split on minor paraphrases, restatements, or
↪  rewordings.
- Do NOT split continuous arithmetic or symbolic chains
↪  unless clearly interrupted by a change in epistemic
↪  role.
- DO split when the epistemic role changes, even subtly.
- When in doubt → do NOT split (prefer fewer, larger
↪  segments).
Decoding guidance:
- Only emit a segment token if starting a new reasoning
↪  step.
- Otherwise, continue the text verbatim without
↪  modification.
```

system_prompt_surprisal.

```
You are a reasoning trace segmenter.
You will be given a reasoning trace.
You must reproduce the trace EXACTLY as given, token by
↪  token, without paraphrasing, omission, or reordering.
While reproducing the trace, internally identify valid
↪  reasoning segment boundaries.
A segment boundary is a point between two sentences where
↪  the next sentence begins a new or different epistemic
↪  function.
You are NOT required to output boundaries or modify the
↪  text in any way.
Your only visible output must be the exact original trace.
Rules:
- Each segment corresponds to one coherent reasoning step
↪  (a single epistemic function).
- Boundaries are allowed ONLY between complete sentences.
- NEVER consider boundaries inside a sentence.
- A boundary is valid ONLY when the next sentence begins a
↪  new or different epistemic function.
- Do NOT judge correctness, logic, or factual accuracy —
↪  focus only on change of epistemic role.
- Boundaries may exist even when topic, entities, or
↪  vocabulary remain similar.
Main epistemic functions (reference only — do NOT label):
1. Assumption / Setup (premises, constraints, known facts)
2. Intermediate Inference (deriving new information)
3. Contrast / Objection / Alternative
4. Revision / Correction / Reconsideration
5. Goal / Plan / Next-step management
6. Conclusion / Final answer / Summary
Hard constraints:
- Boundaries may only occur between complete sentences.
- Segments are contiguous and non-overlapping.
- Every character of the text belongs to exactly one
↪  segment.
Granularity:
- Do NOT split on minor paraphrases, restatements, or
↪  rewordings.
- Do NOT split continuous arithmetic or symbolic chains
↪  unless clearly interrupted by a role change.
- DO recognize a boundary when the epistemic role changes,
↪  even subtly.
- When in doubt → do NOT split (prefer fewer, larger
↪  segments).
```

D.2 Schema-Labeling Prompts

system_prompt_reasoning_flow.

```
You are a reasoning trace classifier. You are given a
↪  target text snippet along with its surrounding context:
↪  one or more previous snippets and one or more next
↪  snippets from a reasoning trace. Your task is to
↪  classify the TARGET SNIPPET into EXACTLY ONE label from
↪  the reasoning trace schema below, using the context to
↪  help disambiguate.
** Labels and Definitions:
- Context: Background information or problem statement.
↪  Given before reasoning begins.
- Planning: High-level or stepwise plan for reasoning.
↪  Includes verbs like check, solve, start by, find.
- Fact: Objective knowledge, rules, constants, theorems, or
↪  world knowledge not specific to this problem.
- Reasoning: Logical deduction, calculation, simplification,
↪  or inference. Applying facts to the problem.
- Restatement: Rephrasing or clarifying previous text,
↪  without adding new reasoning.
- Assumption: Explicit or implicit assumptions used for
↪  reasoning, including branching or hypothetical cases.
- Example: Specific instance or illustration of a general
↪  concept or pattern.
- Reflection: Subjective commentary, doubt, confidence, or
↪  meta-reasoning.
- Conclusion: Final answer or judgment. Should represent
↪  the resolved result of reasoning.
** Rules for Classification:
- Each TARGET SNIPPET gets EXACTLY ONE label.
- Focus on the intent and function of the snippet in
↪  reasoning, using previous and next snippets to provide
↪  context.
- If the snippet mixes multiple types, classify based on
↪  the PRIMARY function.
- Context snippets are only for disambiguation and should
↪  not themselves be labeled.
** Input Format:
Previous Context: "<one or more previous snippets>"
Target Snippet: "<snippet to classify>"
Next Context: "<one or more next snippets>"
** Output Format:
Output: {"label": "Label from schema"}
** Example:
Previous Context: "I need to find the sum of all integer
↪  bases b > 9."
Target Snippet: "Let me write that division as a fraction:
↪  (9b + 7)/(b + 7)."
Next Context: "Maybe I can simplify this expression."
Output: {"label": "Reasoning"}
```

user_prompt_reasoning_flow.

```
# Task
Analyze the following triplet and provide only label for
↪  the **Target Segment**, nothing else.
**Input Data:**
Previous Context:* "{PREVIOUS_SEGMENT}"
Target Snippet:* "{TARGET_SEGMENT}"
Next Context:* "{NEXT_SEGMENT}"
Output:
```

system_prompt_argument.

```
You are an expert annotator for Argument Mining on
↪  Chain-of-Thought (CoT) reasoning traces. Your task is
↪  to classify the role of a specific **Target Segment**
↪  within a reasoning process using a specific 5-label
↪  schema.
You will be provided with:
1. **Previous Context:** The segment immediately preceding
↪  the target.
2. **Target Segment:** The specific text you must classify.
3. **Next Context:** The segment immediately following the
↪  target.
# Classification Decision Tree
To determine the label, ask the following questions about
↪  the **Target Segment**:
1. **Is it a given fact, rule, or definition?**
   → Label: **[PREMISE]**
   * *Includes:* Math rules, world knowledge, stated
   ↪  problem constraints, or calculation inputs.
2. **Is it a derivation, hypothesis, or result derived from
↪  a previous step (but not the final answer)?**
   → Label: **[INTERMEDIATE CLAIM]**
   * *Includes:* Simplifications, intermediate calculations,
   ↪  temporary assumptions ("Maybe it is...").
3. **Does it negate, refine, or correct a previous
↪  thought?**
   → Label: **[REBUTTAL/CORRECTION]**
   * *Includes:* Self-corrections ("Wait, no"), realization
   ↪  of errors, or rejecting a hypothesis based on
   ↪  constraints.
4. **Is it the final output or solution for the user?**
   → Label: **[FINAL CLAIM]**
   * *Includes:* The final answer statement, the concluding
   ↪  sentence of the entire trace.
5. **Is it none of the above (purely planning or
↪  structural)?**
   → Label: **[META]**
   * *Includes:* Planning ("Let's try..."), navigation
   ↪  ("Let me check"), or stating goals.
# Examples
**Example 1**
*Previous Context:* "I need to find the sum of all integer
↪  bases b > 9."
*Target Segment:* "Let me write that division as a
↪  fraction: (9b + 7)/(b + 7)."
*Next Context:* "Maybe I can simplify this expression."
*Output:* {"label": "META"}
**Example 2**
*Previous Context:* "So 9b +7 = 9*(b +7) - 56."
*Target Segment:* "Therefore, (9b +7)/(b +7) = 9 - 56/(b
↪  +7)."
*Next Context:* "For this to be an integer, 56/(b +7) must
↪  be an integer."
*Output:* {"label": "INTERMEDIATE CLAIM"}
**Example 3**
*Previous Context:* "14 is 14, which would make b +7=14 →
↪  b=7."
*Target Segment:* "But b must be greater than 9. So 14 is
↪  too small."
*Next Context:* "Similarly, 8 and 7 are too small."
*Output:* {"label": "REBUTTAL/CORRECTION"}
**Example 4**
*Previous Context:* "So the answer is 21 +49=70."
*Target Segment:* "Therefore, the final answer is 70."
*Next Context:* "[END OF TRACE]"
*Output:* {"label": "FINAL CLAIM"}
```

user_prompt_argument.

```
# Task
Analyze the following triplet and provide only label for
↪  the **Target Segment** in JSON format, nothing else.
**Input Data:**
*Previous Context:* "{PREVIOUS_SEGMENT}"
*Target Segment:* "{TARGET_SEGMENT}"
*Next Context:* "{NEXT_SEGMENT}"
*Output:*
```

system_prompt_thought_anchor.

```
You are an expert annotator for Large Language Model
↪  reasoning traces. Your task is to classify the
↪  functional role of a specific **Target Segment** within
↪  a reasoning process using a specific 8-label schema
↪  derived from the Thought Anchor (Bogdan et al. 2025)
↪  extension to the Venhoff et al. (2025) framework.
You will be provided with:
1. **Previous Context:** The segment immediately preceding
↪  the target.
2. **Target Segment:** The specific text you must classify.
3. **Next Context:** The segment immediately following the
↪  target.
# Classification Decision Tree
To determine the label, analyze the **Target Segment** in
↪  relation to the context and select the first category
↪  that applies from the list below:
1. **Is it explicitly stating the final solution?**
   → Label: **[Final Answer Emission]**
   * *Includes:* "Therefore, the answer is X," or the boxed
   ↪  solution.
2. **Is the model expressing confusion, doubting a previous
↪  step, or deciding to backtrack?**
   → Label: **[Uncertainty Management]**
   * *Includes:* "Wait, that can't be right," "I am
   ↪  confused," "Let me re-read the question," or "Let's
   ↪  go back to the start."
3. **Is the model verifying a previous step or
↪  double-checking a calculation?**
   → Label: **[Self Checking]**
   * *Includes:* "Let me verify that calculation,"
   ↪  "Checking if this fits the constraints," or
   ↪  re-calculating to confirm a result.
4. **Is the model parsing the input, restating the question,
↪  or defining variables based on the prompt?**
   → Label: **[Problem Setup]**
   * *Includes:* "We are looking for X," "Let x be the
   ↪  number of apples," or rephrasing the user's request.
5. **Is the model proposing a strategy, stating a roadmap,
↪  or engaging in meta-reasoning about *how* to solve it?**
   → Label: **[Plan Generation]**
   * *Includes:* "I will use the Pythagorean theorem,"
   ↪  "Let's break this into two cases," or "First I will
   ↪  find X, then Y."
6. **Is the model recalling a static fact, formula, or
↪  specific problem detail *without* performing new
↪  derivation?**
   → Label: **[Fact Retrieval]**
   * *Includes:* "The derivative of sin(x) is cos(x)," "The
   ↪  problem states that the car implies 50mph," or
   ↪  recalling a constant like Pi.
7. **Is the model performing algebraic manipulation,
↪  logical deduction, or calculating new values?**
   → Label: **[Active Computation]**
   * *Includes:* "5 * 12 = 60," "Simplifying the equation
   ↪  gives x = 2," or applying a logical rule to derive a
   ↪  new state.
8. **Is the model summarizing intermediate findings or
↪  aggregating results from different steps?**
   → Label: **[Result Consolidation]**
   * *Includes:* "So, from Case 1 we have 5, and from Case
   ↪  2 we have 10," or "Combining these equations..."
# Examples
**Example 1**
*Previous Context:* "The problem asks for the area of the
↪  circle."
*Target Segment:* "First, I need to find the radius, and
↪  then I can apply the area formula."
*Next Context:* "The diameter is given as 10 units."
*Output:* {"label": "Plan Generation"}
**Example 2**
*Previous Context:* "We have the equation 2x + 4 = 12."
*Target Segment:* "Subtracting 4 from both sides, we get 2x
↪  = 8, so x = 4."
*Next Context:* "Now let's check if this satisfies the
↪  original condition."
*Output:* {"label": "Active Computation"}
**Example 3**
*Previous Context:* "So x = 4."
*Target Segment:* "Wait, if x is 4, then the denominator
↪  would be zero. That's impossible."
*Next Context:* "I must have made a mistake in the
↪  factoring step."
*Output:* {"label": "Uncertainty Management"}
**Example 4**
*Previous Context:* "We need the atomic weight of Carbon."
*Target Segment:* "Carbon has an atomic weight of
↪  approximately 12.011."
*Next Context:* "So the total molar mass is..."
*Output:* {"label": "Fact Retrieval"}
**Example 5**
*Previous Context:* "Case A yielded 4 possibilities. Case B
↪  yielded 2."
*Target Segment:* "So, in total, we have 4 + 2 = 6
↪  possibilities so far."
*Next Context:* "Now, let's look at the final answer."
*Output:* {"label": "Result Consolidation"}
**Example 6**
*Previous Context:* "The calculation gave 104."
*Target Segment:* "Let me quickly multiply 13 by 8 again to
↪  be sure... yes, it is 104."
*Next Context:* "So the value is correct."
*Output:* {"label": "Self Checking"}
```

user_prompt_thought_anchor.

```
Analyze the following triplet and provide only label for
↪  the **Target Segment** in JSON format, nothing else.
**Previous Context:** {PREVIOUS_SEGMENT}
**Target Segment:** {TARGET_SEGMENT}
**Next Context:** {NEXT_SEGMENT}
*Output:*
```

D.3 Topic-Label Prompt

system_prompt_topic_label.

```
You are an AI assistant tasked with labeling clusters of
↪  text segments extracted from the internal reasoning
↪  trace of a large language model (LLM).
Each cluster contains multiple text segments that:
- May be non-contiguous
- Come from the LLM's step-by-step reasoning
- Represent part of how the model is thinking, not just
↪  what it is talking about
Your task is NOT to label the surface topic or semantic
↪  subject matter.
Instead, you must infer the LLM's reasoning role or
↪  cognitive operation represented by the cluster.
Focus on identifying what the model is doing, such as:
- Reframing or restructuring the problem
- Introducing substitutions or abstractions
- Applying logical or causal rules
- Justifying a methodological choice
- Reducing a complex condition to simpler cases
- Checking constraints or edge cases
- Interpreting intermediate results
When assigning a label, ask yourself:
- What reasoning function do these segments serve in the
↪  solution process?
- How do they move the reasoning forward?
Your output must be:
- A short, concise label (1–10 words)
- Describing the reasoning process, not the content
- With no additional explanation or text
---
Example 1
Segments:
- "The anti-flag antibody is included to confirm that the
↪  MUC1 construct is properly expressed and presented on
↪  the cell surface, independent of glycan modification."
- "Including the anti-flag allows detection of MUC1
↪  expression and ensures that any changes in binding are
↪  due to the sugar modification rather than altered
↪  protein expression."
- "The anti-flag antibody should be added in parallel with
↪  the primary detection steps to serve as a control."
Correct label:
Justifying experimental controls in assay design
---
Example 2
Segments:
- "Let me note that n + 3 is (n + 2) + 1. Maybe I can use
↪  substitution. Let me set k = n + 2."
- "After substitution, the product becomes 3(k + 1)(k² -4k
↪  +13)."
```

- "Since k and k + 1 are coprime, k must divide 3(k² −4k
↪ +13)."
- "This reduces the condition to k dividing 39."
Correct label:
Reducing divisibility constraints via substitution and
↪ coprimality