TEXT2STORIES: Evaluating the Alignment Between Stakeholder Interviews and User Stories

Anonymous ACL submission

Abstract

Software requirements are derived from natural language inputs such as the transcripts of elicitation interviews. However, evaluating whether those derived requirements faithfully reflect the stakeholders' needs remains a challenging manual task. We introduce TEXT2STORIES, a task and metrics for text-to-story alignment that allow quantifying the extent to which requirements (in the form of user stories) match the actual needs expressed by the elicitation session participants. Given an interview transcript and a set of user stories, our metric quantifies (i) *correctness*: the proportion of stories supported by the transcript, and (ii) *completeness*: the proportion of transcript supported by at least one story. We segment the transcript into text chunks and instantiate the alignment as a matching problem between chunks and stories. Experiments over four datasets show that an LLM-based matcher achieves 0.86 macro-F1 on manually labeled chunk–story pairs, while embedding models alone remain behind but enable effective blocking. Automated alignment decisions enable the downstream computation of *completeness* and *correctness*. We show that these two metrics behave consistently with human judgment across multiple datasets, positioning TEXT2STORIES as a scalable, source-faithful complement to existing user-story quality criteria.

1 Introduction

User stories are a key asset in agile development: short, semi-structured requirements that capture *who* wants *what* and *why* (Cohn, 2004).[1] In practice, requirements such as user stories are most commonly distilled *manually* from lengthy, conversational sources such as stakeholder interviews and facilitated workshops (Wagner et al., 2019).

Large language models (LLMs) promise to streamline this tedious human activity by generating candidate user stories from unstructured interview transcripts (Ronanki et al., 2022; Quattrocchi et al., 2025; Yamani et al., 2025).

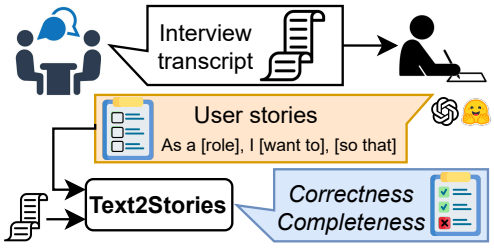

Figure 1: Overview of the workflow. Elicitation from stakeholders produces an interview transcript. From this transcript, human analysts (or LLMs) craft user stories. Given a set of stories and the original transcript, TEXT2STORIES computes two metrics: *Correctness* (are stories supported by the interview?) and *Completeness* (is the whole conversation covered by the stories?).

Yet, a central question remains largely unaddressed: *to what extent do the generated stories faithfully reflect what stakeholders actually said?* This question applies to both human- and LLM-generated user stories. Existing quality frameworks for user stories, like QUS (Lucassen et al., 2016), only assess how well a story is written, not whether it is *grounded in the source*. Addressing this question is critical, because hidden and incomplete requirements are the most frequent cause for software project failure (Fernández et al., 2017).

We address this gap by introducing TEXT2STORIES, a task and metric for *text-to-story alignment (T2SA)*. The goal is to quantify alignment between an interview transcript and a set of user stories derived from it. Rather than scoring stories in isolation, T2SA evaluates them *against the source*, offering a complementary perspective to traditional user-story quality checks. We use alignment for two interpretable, source-faithful measures: *correctness*, the proportion of stories supported by at least one segment of the transcript, and *completeness*, the proportion of transcript

[1] We consider user stories following the Connextra template: "*As a [role], I want to [action], so that [benefit]*".

Figure 2: Example of Text-to-Story Alignment (T2SA). A snippet from the elicitation interview (left) is aligned against two candidate user stories by a *human analyst* (right). One is a **valid match** (approval of new teams is evidenced in the interview), the other is **not a match** (allocation of match officials is not supported in this excerpt). T2SA produces traceable chunk-story links used by our Correctness and Completeness metrics.

segments that are covered by at least one story. As depicted in Figure 1, given an interview's transcript and the set of corresponding stories, the goal is to output the two measures for an actionable quality assessment of the stories at hand.

We cast T2SA as a matching problem between conversation transcript *chunks* and stories, as depicted in Figure 2. Segmenting interviews into overlapping chunks enables local grounding and traceability: each positive match points to concrete evidence and can be surfaced as a justification during review (Spijkman et al., 2022). However, naively evaluating all story-chunk pairs scales quadratically and can become cost prohibitive for long interviews or large story sets. To make T2SA practical, we introduce an *embedding-based blocking* scheme that preserves alignment quality while reducing LLM token processing, making source-grounded evaluation feasible (Papadakis et al., 2020). We report the following contributions.

(1) We formalize the *text-to-story alignment (T2SA)* task and introduce two complementary, source-faithful measures, *correctness* and *completeness*, that quantify how well stories are grounded in interview transcripts (Section 3).

(2) We instantiate T2SA as pairwise matching between transcript chunks and stories, with interchangeable matchers (embeddings, cross-encoders, LLM judges) and an embedding-based blocking operator (Section 4).

(3) We conduct experiments on 17 software projects (Section 5) and show that an LLM-based matcher achieves an average macro-F1 of 0.86 on four annotated datasets for the pairwise matching task (Section 6.1). Using the top-performing matcher and the non-annotated datasets, we demonstrate that *correctness* and *completeness* can effectively compare generated user stories (both human- and LLM-written) (Section 6.2). We also show that blocking reduces token processing up to 3× (Section 6.3) and that chunking leads to almost doubling the average macro-F1 in matching, compared to using the full document in the task (Section 6.4).

2 Related Work

TEXT2STORIES turns a common but under-formalized software-engineering activity into an NLP task, involving methods from different research areas.

Requirements engineering (RE). User stories are central to agile RE, and their textual quality is typically assessed with frameworks such as QUS (Lucassen et al., 2016). While valuable, these criteria target how well a story is written (syntax, semantics, pragmatics) rather than whether the story is *grounded* in the elicitation source. Closer to our goal, Spijkman et al. link user stories to elicitation conversations (Spijkman et al., 2022) and study how to summarize conversation transcripts to surface requirements-relevant turns (Spijkman et al., 2023). TEXT2STORIES builds on this work by (i) formalizing alignment *between* stories and interview chunks and (ii) turning grounding into explicit, coverage-style metrics (correctness/completeness) rather than heuristics on isolated stories.

Faithfulness and factuality. Existing work measures whether generated text is faithful to its source. In summarization, hallucinations motivate source-grounded evaluation (Maynez et al., 2020; Pagnoni et al., 2021; Nenkova and Passonneau, 2004); automatic metrics rely on NLI or QA signals to score consistency (Kryściński et al., 2020; Laban et al., 2022; Wang et al., 2020; Scialom et al., 2021; Fabbri et al., 2022; Zha et al., 2023). TEXT2STORIES is aligned with these lines (judge against the source), but differs in unit and objective: we align *requirements* to *conversational* evidence with many-to-many relations, and we report interpretable set-level coverage measures.

Work on fact verification formalizes support/contradiction with retrieved evidence (Thorne et al., 2018) and detects previously fact-checked

claims with matching (Shaar et al., 2020). Our setting differs as (i) claims (user stories) are longer, compositional artifacts rather than atomic statements, and (ii) evidence lives in long, multi-speaker transcripts with local context and discourse.

Retrieval, blocking, and reranking. Our *embedding-based blocking* and *pairwise matching* mirror standard IR practice: bi-encoders for fast similarity search (Reimers and Gurevych, 2019), cross-encoder rerankers for precise scoring (Nogueira and Cho, 2019), and techniques for reducing quadratic comparisons (Papadakis et al., 2020). Unlike open-domain retrieval, our goal is not answering a query but preserving alignment *recall* at minimal token cost so that a stronger judge can operate on a pruned set of story–chunk pairs.

LLM-as-a-judge. There is evidence of strong agreement of LLMs and human preferences in grading outputs, with growing understanding of biases and mitigation (Zheng et al., 2023). TEXT2STORIES leverages LLM-as-a-judge constrained to *pairwise* decisions at the chunk level. No prior work operationalizes a *text-to-story* alignment task with correctness/completeness.

3 Task: Text-to-Story Alignment (T2SA)

We define the **text-to-story alignment (T2SA)** task as follows: given a transcript of an elicitation interview and a set of user stories derived from it, the goal is to quantify the extent to which the stories accurately reflect the information expressed by the participants during the interview.

Formally, let T be a transcript segmented into n text chunks $C = \{c_1, c_2, \ldots, c_n\}$, and let $S = \{s_1, s_2, \ldots, s_m\}$ be a set of m user stories. The goal is to determine, for each pair (c_i, s_j), whether the chunk c_i *supports* the user story s_j, i.e., whether the requirement expressed in s_j can be justified by information contained in c_i. We define two metrics:

Correctness: proportion of user stories supported by at least one chunk:

$$\text{Corr} = \frac{\left|\{\, s_j \in S : \exists\, c_i \in C,\ y(c_i, s_j) = 1 \,\}\right|}{|S|}, \tag{1}$$

where $y(c_i, s_j) \in \{0, 1\}$ is the *alignment label*, with 1 if the requirement in s_j is justified by information in c_i, 0 otherwise.

Completeness: proportion of chunks in the transcript that are covered by at least one story:

$$\text{Comp} = \frac{\left|\{\, c_i \in C : \exists\, s_j \in S,\ y(c_i, s_j) = 1 \,\}\right|}{|C|}. \tag{2}$$

Completeness is a relative metric and may not reach 1.0, since not all transcript chunks necessarily express stakeholder needs. It measures transcript coverage rather than true requirement completeness; however, due to the lack of gold-standard requirement catalogs and the high cost of creating them, we use it as a proxy. A small human audit (Appendix B.5) shows that sets with more requirements also achieve higher completeness, providing empirical evidence of a positive correlation between transcript coverage and requirement completeness in our setting.

Since interviews vary widely in length and structure, these metrics are intended for *within-interview comparisons*, such as comparing alternative story sets derived from the same transcript.

4 Method

We define our method to solve the T2SA task as a two-stage pipeline: (1) segmenting the transcript into chunks, and (2) aligning the chunks with user stories via a matcher. An optional blocking stage can be inserted between these steps to reduce computational cost without altering the task definition. Correctness and completeness measures are then derived from the alignment output.

Formally, let $C = \{c_1, \ldots, c_n\}$ be the set of transcript chunks and $S = \{s_1, \ldots, s_m\}$ the set of user stories. We consider the set of candidate pairs

$$P = C \times S = \{(c_i, s_j) \mid c_i \in C, s_j \in S\}.$$

A matcher X is a function

$$X : C \times S \to \{0, 1\},$$

where $X(c_i, s_j) = 1$ indicates that chunk c_i *supports* story s_j. By default, X is applied to the entire Cartesian product P, i.e., all possible pairs.

The overall method admits two operating modes:
(1) **Direct matching**: evaluate X on all pairs $P = C \times S$.
(2) **Blocked matching**: apply an embedding-based blocking operator B_K that restricts P to a smaller candidate set P', then evaluate X only on P'.

The following subsections detail the three core components: *chunking* (C), *pairwise matching* (X), and *embedding-based blocking* (B_K).

4.1 Chunking (C)

We segment transcripts based on speaker turns (as in Figure 2), following prior work on identifying requirements information in interviews (Spijkman et al., 2023). While one could in principle treat each turn as a separate chunk, elicitation interviews are a question and answer sequence (Zaremba and Liaskos, 2021), which provides contextual information (spanning across multiple turns) that helps understand whether a user story is supported. Following expert knowledge (Spijkman et al., 2023), we adopt the strategy of grouping three consecutive speaker turns into a unit, using a stride of one so that every possible three-turn window is considered and full transcript coverage is ensured. Given turns $(t_1, t_2, \ldots, t_k)$, this results in overlapping chunks $(t_1, t_2, t_3), (t_2, t_3, t_4), \ldots, (t_{k-2}, t_{k-1}, t_k)$. In principle, the chunking strategy is flexible, however, a full sweep of chunk sizes would require manually annotating alignments for each variant, which is prohibitively expensive, so we conduct our experiments with the three-turn sliding window approach. Our ablation shows that matching stories against the entire interview document leads to substantially degraded alignment performance compared to matching against chunks (Section 6.4).

4.2 Pairwise Matching (X)

Given the chunk set C and the user stories S, we form all possible pairs (c_i, s_j). For each pair, a model X assigns a binary label indicating whether the chunk supports the story.

The matcher X can be instantiated as:
(1) **LLM based judge**, a large language model prompted with few shot examples of aligned and non aligned pairs (prompts in Appendix A.2);
(2) **Cross encoder**, a reranker model that jointly encodes (c_i, s_j) and outputs a support score; or
(3) **Bi encoder**, which computes embeddings for c_i and s_j independently and checks their cosine similarity.

In all cases, X answers the question: *"Does this interview chunk justify this user story?"*

4.3 Embedding-Based Blocking (B_K)

To reduce computational cost, we introduce an embedding-based blocking step. Instead of evaluating all pairs, we first compute embeddings for all chunks and stories and measure their similarity. For each story s_j, we keep only the Top-K most similar chunks: $B_K(S, C) = \bigcup_{j=1}^{m} \{(c_i, s_j) : c_i \in \text{TopK}(C, s_j)\}$. The matcher X is then applied only to these reduced candidate sets $P' = B_K(S, C)$. All other pairs are implicitly labeled as $X(c_i, s_j) = 0$. This blocking scheme preserves alignment quality while drastically reducing computation by a factor $\frac{|C|}{K}$. For instance, with $|C| = 100$ and $K = 25$, blocking cuts the number of model calls by $4\times$.

5 Experimental Setup

Datasets. We evaluate our approach on 17 datasets (Table 1). The primary source consists of 15 software projects from a requirements engineering course. In each project, students conducted elicitation interviews while role-playing stakeholders and analysts to gather requirements for a target software system. For each project, we collected (i) transcripts of two elicitation interviews, concatenated into a single document, and (ii) the corresponding set of user stories. All data are in English and each project targets a distinct application domain. Students were allowed to refine requirements outside the elicitation sessions. Due to privacy constraints, these datasets cannot be released publicly and only local models can be used on them. In addition, we include two public datasets:
(1) *Public Interview Dataset*: a public interview transcript (Spijkman et al., 2023), combined with a set of user stories (Dalpiaz et al., 2020) for the same target system, forming a transcript–stories pair comparable to our private projects.
(2) *System Description Dataset*: a textual system description with corresponding user stories (Santos et al., 2025). While this dataset does not originate from interviews, it is a valuable test case for validating our method on publicly available data. Consequently, we do not build chunks based on speaker turns, but instead split only at newlines.

To the best of our knowledge, no public dataset explicitly annotates which segments of a natural language source (interview transcripts or descriptions) support individual user stories. Existing work either evaluates user stories in isolation (Yamani et al., 2025; Quattrocchi et al., 2025), or compares generated stories against a reference set, without grounding them in the originating text (Sharma and Kumar Tripathi, 2025; Ronanki et al., 2022). (Korn et al., 2025) mark whether high-level requirements are present in a transcript, but do not identify their exact location. In contrast, our annotated datasets explicitly target *source grounding* by providing manually created alignments between user

Dataset	#Stories	#Chunks	|C×S| tokens	Domain
Student Project 1	53	386	3.76M	Predictive Dental Appointment Scheduler
Student Project 2	44	408	3.26M	Centralized communication system for supermarkets
Student Project 3	59	546	3.87M	Information system for modernization of bakery
Student Project 4	80	169	2.61M	Learning support information system
Student Project 5	50	564	3.78M	Bike business improvement to kickstart efficiency
Student Project 6	62	380	4.73M	Workflow management system for solar panel company
Student Project 7	82	450	5.84M	Movie and series availability system
Student Project 8	46	277	2.52M	User-friendly app for devs to upload apps on Steam
Student Project 9	57	538	4.48M	Unified digital workspace for UX agency
Student Project 10	55	201	2.57M	Information system for automated train operation
Student Project 11	71	406	4.92M	Driving schools comparator
Student Project 12	52	714	4.36M	Cinema organization
Student Project 13	121	386	9.29M	Football league operations department
Student Project 14	64	440	4.49M	Scheduling tool for pizzeria
Student Project 15	39	429	2.39M	Venue management system events coordination
Public Interview	37	132	0.88M	International football association portal
System Description	115	37	0.19M	Travel agency management system

Table 1: Statistics of the 17 datasets. Private datasets consist of student projects (each with two interviews).

Dataset	#Total Pairs	#Positive pairs	#Negative pairs
Student Proj. 1 (S1)	287	109	178
Student Proj. 2 (S2)	236	88	148
Public Int. (PI)	180	74	106
System Desc. (SD)	580	302	278

Table 2: Statistics of the annotated test datasets used for evaluation. Each test set is a subset of the full dataset, selected and balanced via our annotation protocol.

stories and corresponding chunks of the source text.

Annotation Protocol. We manually annotated matching pairs for two private student projects (from different domains) and the public datasets (Table 2). Exhaustively annotating all chunk-story pairs is infeasible due to the $|C| \times |S|$ complexity, so we adopted a similarity-driven protocol:

(1) For each user story, we retrieved source chunks ranked by decreasing embedding similarity;

(2) We selected the top-5 non-overlapping chunks (given stride-based chunking);

(3) If fewer than two positive matchings were found among the top-5, we extended the ranking until at least two positives were identified;

(4) When we knew from prior reading that a specific chunk was the most suitable support, we included it even if it was in the top candidates.

This process led to a balanced yet feasible set of annotated examples, which are used exclusively to evaluate the pairwise chunk–story matching step.

We assessed inter-annotator reliability on a sample using Fleiss' κ with three annotators. The overall agreement is κ=0.470 (*moderate* agreement); additional details are provided in Appendix B.

Metrics. We use the following criteria:

(1) *Macro F1-score*: evaluates the alignment quality of matcher X on the annotated datasets.

(2) *Correctness and Completeness metrics*: assessed in different scenarios to demonstrate the usefulness of our approach.

(3) *Computational cost*: measured as the total number of tokens processed, allowing us to compare blocking and non-blocking approaches.

(4) *Execution time*: recorded on a reference GPU setup for a selected experiment.

Implementation. Our Python code is available at `https://anonymous.4open.science/r/txt2stories-A3FD`, the experiments with the two public datasets can be reproduced end-to-end.

6 Results

(i) We first evaluate the quality of matchers for the pairwise matching task against human-annotated pairs (Section 6.1). (ii) We then show that the proposed correctness and completeness metrics, computed over the full set of projects with the best matcher from (i), effectively compare different sets of user stories, including both human- and LLM-generated ones (Section 6.2). (iii) Next, we assess the efficiency of the embedding-based blocking strategy for the matching task (Section 6.3). (iv) Finally, we present ablation studies that analyze the impact of key design choices (Section 6.4).

6.1 Pairwise Alignment Quality

We first evaluate the quality of the pairwise matcher X on the annotated datasets, measuring macro F1-

Matcher (X)	Student 1	Student 2	Public Interview	System Description	Average
Bi-encoder [t] (Qwen3-0.6B)	0.424	0.593	0.649	0.718	0.596
Bi-encoder [t] (Qwen3-4B)	0.362	0.576	0.741	0.741	0.605
Bi-encoder [t] (Qwen3-8B)	0.557	0.635	0.771	0.707	0.668
Cross-encoder [t] (Qwen3-Reranker-0.6B)	0.647	0.679	0.748	0.841	0.729
Cross-encoder [t] (Qwen3-Reranker-4B)	0.765	0.728	0.797	**0.914**	0.801
Cross-encoder [t] (Qwen3-Reranker-8B)	0.776	0.737	0.829	0.895	0.809
LLM (Qwen3-0.6B)	0.310	0.365	0.291	0.507	0.406
LLM (Qwen3-1.7B)	0.326	0.597	0.483	0.693	0.525
LLM (Qwen3-4B)	0.774	0.676	0.732	0.856	0.760
LLM (Qwen3-8B)	0.774	0.739	0.754	0.902	0.792
LLM (Qwen3-14B)	0.739	0.792	0.814	0.888	0.808
LLM (Qwen3-32B)	0.803	0.831	0.815	**0.914**	0.841
LLM (Llama3.2-1B)	0.455	0.418	0.493	0.324	0.423
LLM (Llama3.2-3B)	0.533	0.671	0.601	0.752	0.639
LLM (Llama3.1-8B)	0.578	0.696	0.703	0.863	0.711
LLM (Llama3.3-70B)	**0.818**	**0.840**	**0.876**	0.902	**0.859**

Table 3: Macro F1-scores of different matcher instantiations X on the four annotated datasets. Results for threshold-based models are reported with best threshold and marked with [t].

score across positive and negative pairs. This isolates the second step of our method (alignment). Table 3 reports the results for three matcher families: LLM judges, cross-encoders, and bi-encoders. All experiments are run with LLMs' temperature set to 0. LLM judges emerge as the best-performing matchers. Qwen3-32B and Llama3.3-70B achieve macro-F1 scores above 0.80 across all datasets, with top average scores of 0.841 and 0.859, respectively. Cross-encoders deliver competitive results and are particularly effective at smaller model sizes; however, they output continuous scores and therefore require calibration. Results in Table 3 correspond to the best threshold per dataset, and should be interpreted as optimistic upper bounds (results with calibrated thresholds are reported in Appendix B.1). This reduces their practical appeal, as the marginal gains over same-size LLM judges may not justify the added complexity of threshold calibration. Finally, bi-encoder matchers perform worse overall. with low macro-F1 scores.

Takeaway. *LLM-based judges provide the most reliable alignment decisions for text-to-story matching*, consistently achieving F1-scores above 0.8 and making them well suited for computing downstream correctness and completeness.

6.2 Stories' Correctness and Completeness

We now assess the behavior of our *Correctness* and *Completeness* metrics to validate them as indicators of source alignment. To this end, we examine whether completeness and correctness accurately reflects the well-known external signals that a sound metric should capture. For each of the 15 private student projects, we let Qwen models of increasing size (0.6B to 32B) generate up to 50 user stories from the interview transcripts - the cap prevents excessively long generations and mitigates hallucinated "never-ending" outputs (see Appendix for the generation prompt.). We then apply our TEXT2STORIESpipeline in direct matching mode (no B_K), using Qwen3-32B as matcher X, and compute correctness and completeness. Figures 3 and 4 report the average scores across the 15 datasets, with standard deviation as error bars. We also report the metrics for the human-written stories produced by the students (*Human*). For completeness, we additionally report the average ranking (lower is better) across datasets.

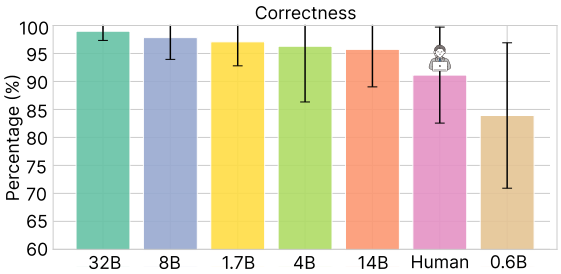

Figure 3: Average *correctness* of user stories generated by Qwen models of increasing size. Error bars denote ± 1 standard deviation across datasets (n=15).

Completeness increases with generator size under a fixed judge and pipeline. If completeness is a sound proxy for coverage, this trend is expected: larger LLMs are known to generate more diverse and exhaustive outputs and should therefore cover a larger fraction of the interview content (Kaplan et al., 2020). The monotonic increase across model

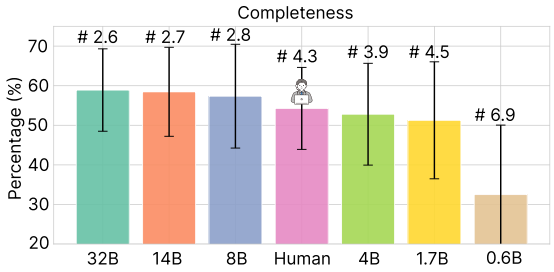

Figure 4: Average *completeness* and model ranking (#) of user stories generated by Qwen models of increasing size. Error bars denote ± 1 standard deviation across datasets (n=15).

sizes suggests that the metric is sensitive to this external signal, a pattern that is also reflected by the average completeness ranking across datasets reported in Table 4. LLM-generated story sets also achieve higher completeness than the human ones. As detailed in Appendix B.5, a targeted audit shows that LLM sets introduce additional (human-validated) functionalities, which explains the higher transcript coverage.

In contrast, **correctness is consistently high** across all but the smallest generator (Qwen3-0.6B). This asymmetry mirrors real-world software engineering practice: elicited requirements are often individually valid, but achieving *completeness* of the requirements set is a recurring challenge due to the implicit and *tacit* nature of requirements (Sutcliffe and Sawyer, 2013; Ferrari et al., 2016). Interestingly, human-written stories obtain lower correctness than most LLM-generated sets. This is expected in our setting, as students were allowed to rely on sources beyond the interview transcripts, leading to stories not grounded in the elicitation data. In Section 6.4, we further demonstrate that both metrics decrease significantly in case of semantic misalignment.

Takeaway. With a fixed matcher and different human/LLM generators for the stories, completeness correlates with generator capacity and the number of human-validated functionalities, while correctness penalizes reliance on information absent from the interview, suggesting that *both metrics behave in an intuitive and human-aligned way*.

6.3 Blocking Efficiency

We evaluate the embedding-based blocking operator (B_K) in terms of reducing computational cost. In particular, we aim to obtain the same pairwise matching performance that would be achieved by applying a matcher to the full $|C| \times |S|$ Cartesian product, while processing fewer tokens. Blocking must retain the pairs classified as positive by the matcher on the full cartesian product; if all positive pairs are preserved, downstream correctness and completeness scores remain unchanged. We use Qwen3-32B as the matcher to compute the set of positive pairs from the full Cartesian product and we compare off-the-shelf Qwen3-Embedding-0.6B and its fine-tuned version. To fine-tune the embedding model, we generate training data by labeling all chunk-story pairs with the matcher and balancing positives and negatives via random downsampling. Evaluation follows a leave-one-out protocol across the 15 private projects.

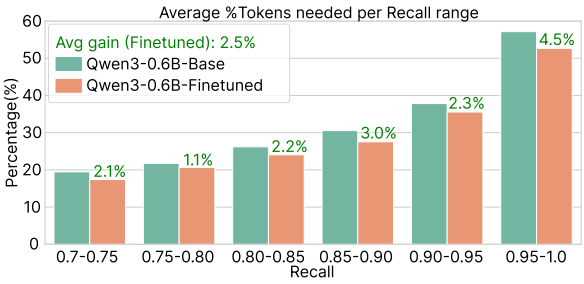

Figure 5: Recall of positive pairs vs. percentage of tokens retrieved by the blocking operator (B_K) across datasets. Lower values indicate higher efficiency.

Figure 5 reports, for increasing recall levels, the fraction of tokens that must be processed after blocking. For each story, we retrieve the top-K most similar chunks and increase K until the desired recall with respect to the reference positives is reached. Because chunk lengths vary, we report token counts rather than K, expressed as a percentage of the full Cartesian product. Blocking is efficient: on average, only one third of the tokens ($3\times$ reduction) are needed for recall 0.9–0.95, and about half ($2\times$ reduction) for recall 0.95–1. Fine-tuning further improves efficiency. In other words, with a $2\times$ reduction in token usage, we obtain nearly the same set of matching pairs as those produced by applying Qwen3-32B to the full Cartesian product.

To complement token-based metrics, we measure runtime on one dataset (Student Project 12). Blocking reduces runtime by almost a factor of 4, from 1.2k seconds to 329 (using 22% of the tokens). Details in Appendix B.

Takeaway. Embedding-based blocking preserves alignment quality while *substantially reducing computational cost*, with the same matching outcomes as exhaustive evaluation.

6.4 Ablations and Sensitivity

Chunking Alignment Quality. Chunking is key to our alignment task, where the matching is between a fragment and a user story. To evaluate its impact, we compare this pairwise approach with an ablation setting in which the model receives the *entire interview transcript*, segmented into numbered chunks (full-context). The model is asked, given a user story, to predict the indices of the supporting chunks. Transcripts are split into batches of 200 chunks to remain within context window limits.

Dataset	S1	S2	PI	SD	Avg
Full Context	0.471	0.540	0.623	0.322	0.489
Pairwise	**0.803**	**0.831**	**0.815**	**0.914**	**0.841**

Table 4: Macro F1-scores comparing full-context evaluation with the pairwise matcher on the annotated datasets (S1, S2, PI, SD). Qwen3-32B is used as matcher X.

As shown in Table 4, providing the model with full-context chunking degrades macro F1-score performance w.r.t. the pairwise approach.

Chunking and Blocking Efficiency. We study how chunking affects the efficiency of the embedding-based blocking operator B_K. Beyond our default setting of three speaker turns (stride 1), we evaluate two-turn speaker chunks and token-based chunking. Table 5 reports the average fraction of tokens retrieved to achieve recall 0.9 and 0.95. Speaker-based chunking is consistently more

Chunking	Recall 0.9	Recall 0.95
Speaker turns (3, stride 1)	35.5%	44.6%
Speaker turns (2, stride 1)	**31.4%**	**40.1%**
Token chunks (200,stride 100)	39.8%	51.3%
Token chunks (400,stride 200)	61.2%	72.9%

Table 5: Blocking efficiency under different chunking strategies: average fraction of tokens retrieved by B_K across datasets to achieve recall 0.9 and 0.95. Lower is better. (Qwen3-Embedding-0.6B-Base)

efficient than token-based chunking, likely because speaker turns form semantically coherent units aligned with conversational structure, while token-based splits often truncate discussions mid-turn. Using shorter speaker-based windows (two turns) improves efficiency further, although this may come at the cost of reduced context per chunk.

Robustness of the Metric. To assess robustness, we evaluate *Correctness* and *Completeness* when user stories are deliberately mismatched with unrelated transcripts (e.g., swapping stories between the two public datasets). Table 6 shows the results for the public datasets. As expected, both metrics decrease significantly in the swapped setting, confirming that TEXT2STORIES is sensitive to semantic misalignment.

Metric	PI	SI	PI Stories-SD	SD Stories-PI
Corr	94.6	99.1	2.7	6.9
Comp	59.1	72.9	5.4	3.0

Table 6: Correctness and completeness under in-domain and swapped for the public datasets (PI, SI).

Model	32B	8B	14B	H	4B	1.7B	0.6B
Comp (%)	49.1	48.9	48.5	46.9	44.2	42.7	24.6
Avg Rank (#)	2.5	2.4	2.4	3.8	3.6	4.0	5.9

Table 7: Completeness and average rank after collapsing overlapping matching chunks (one shared speaker turn).

Completeness inflation. Completeness can be inflated by overlapping chunks (stride = 1), as a requirement expressed in a single turn may appear in multiple windows. However, the metric is intended for *within-interview* comparisons (e.g., human vs. LLM-generated stories for the same transcript), where interview structure and chunking are fixed and therefore affect all systems equally. To quantify this effect, we recomputed completeness after retaining only matching chunks that overlap by at most one speaker turn. The resulting ranking (Table 7) is almost identical to the original setup (Figure 4), indicating that overlap-induced inflation does not affect conclusions in our use case.

7 Conclusion

We introduce TEXT2STORIES, a source-grounded evaluation framework that formalizes *text-to-story alignment* as many-to-many matching between interview chunks and user stories. Our pipeline combines turn-aware chunking, a calibration-free pairwise judge, and an embedding-based blocking operator that preserves alignment recall while reducing token cost. On four annotated sets, LLM judges attain strong alignment quality enabling the computation of two quality measures: Correctness and Completeness. Applying these metrics to generated stories show that they effectively assess both human- and LLM-written user stories, producing results aligned with human judgments.

Limitations

Our *completeness* measures transcript coverage rather than the harder notion of requirements completeness. Our preliminary audit (Appendix B) suggests that LLM-generated stories introduce additional functionalities beyond the human sets, hinting at potential gains in requirements completeness. However, a rigorous correlation study aligning transcript coverage with human expert-validated requirement catalogs would be valuable to establish the metric's effectiveness.

Our metric is not a universal measure of user story quality. TEXT2STORIES focuses on the *source alignment* aspect and should be seen as a complement to existing quality frameworks such as QUS (Lucassen et al., 2016). While QUS criteria capture dimensions such as atomicity, estimability, uniqueness, and other syntactic, semantic and pragmatic criteria, our metric does not evaluate how well a story is written or whether it follows the user story template. Currently, comparison with other metrics is difficult because as there is no baseline for measuring how well a set of user stories reflects the content of stakeholder interviews.

Finally, our evaluation datasets are limited in scope. We rely primarily on student projects, which may not fully reflect industrial elicitation practices. Only two datasets are publicly available, both annotated in terms of matching by one of the authors and cross-checked with the other two on a subset, which may introduce annotation noise. Some experiments cannot be fully reproduced because certain transcripts cannot be released due to confidentiality constraints.

Ethical Considerations

Data from the student projects were collected upon ethical approval from the hosting university. The participating students were asked to grant informed consent and we only use data for which all participating students expressed consent.

Risks. Our metrics estimate *correctness* and *completeness* but do not replace human validation: they can under/over-represent minority stakeholders' needs, so we recommend human-in-the-loop use and discourage deployment in safety-critical settings. LLM judges and embeddings may encode societal biases and introduce evaluation coupling; we therefore report results with multiple models and provide seeds/settings for reproducibility. To reduce leakage, data are processed on local infrastructure.

References

Mike Cohn. 2004. *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional.

Fabiano Dalpiaz, Arnon Sturm, and Patrizia Gieske. 2020. Extraction of conceptual models: User stories vs. use cases. Zenodo dataset.

Alexander Fabbri, Chien-Sheng Wu, Wenhao Liu, and Caiming Xiong. 2022. Qafacteval: Improved qa-based factual consistency evaluation for summarization. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2587–2601, Seattle, United States. Association for Computational Linguistics.

D Méndez Fernández, Stefan Wagner, Marcos Kalinowski, Michael Felderer, Priscilla Mafra, Antonio Vetrò, Tayana Conte, M-T Christiansson, Des Greer, Casper Lassenius, Tomi Männistö, Maleknaz Nayabi, Markku Oivo, Birgit Penzenstadler, Dietmar Pfahl, Guenther Ruhe Rafael Prikladnicki, André Schekelmann, Sagar Sen, Rordrigo Spinola, and 3 others. 2017. Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice. *Empirical Software Engineering*, 22(5):2298–2338.

Alessio Ferrari, Paola Spoletini, and Stefania Gnesi. 2016. Ambiguity and tacit knowledge in requirements elicitation interviews. *Requirements Engineering*, 21(3):333–355.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Alexander Korn, Samuel Gorsch, and Andreas Vogelsang. 2025. LLMREI: Automating requirements elicitation interviews with LLMs. In *Proceedings of the 33rd IEEE International Requirements Engineering conference*.

Wojciech Kryściński, Bryan McCann, Caiming Xiong, and Richard Socher. 2020. Evaluating the factual consistency of abstractive text summarization. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9332–9346, Online. Association for Computational Linguistics.

Philippe Laban, Tobias Schnabel, Paul N. Bennett, and Marti A. Hearst. 2022. SummaC: Re-visiting NLI-based models for inconsistency detection in summarization. *Transactions of the Association for Computational Linguistics*, 10:163–177.

Garm Lucassen, Fabiano Dalpiaz, Jan Martijn E. M. van der Werf, and Sjaak Brinkkemper. 2016. Improving agile requirements: The Quality User Story framework and tool. *Requirements Engineering*, 21:399–417.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On faithfulness and factuality in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1906–1919, Online. Association for Computational Linguistics.

Ani Nenkova and Rebecca Passonneau. 2004. Evaluating content selection in summarization: The pyramid method. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 145–152. Association for Computational Linguistics.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with BERT. *arXiv preprint arXiv:1901.04085*.

Artidoro Pagnoni, Vidhisha Balachandran, and Yulia Tsvetkov. 2021. Understanding factuality in abstractive summarization with FRANK: A benchmark for factuality metrics. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4812–4829, Online. Association for Computational Linguistics.

George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys*, 53(2).

Giovanni Quattrocchi, Liliana Pasquale, Paola Spoletini, and Luciano Baresi. 2025. Can LLMs generate user stories and assess their quality? *arXiv preprint arXiv:2507.15157*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. 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 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Krishna Ronanki, Beatriz Cabrero-Daniel, and Christian Berger. 2022. Chatgpt as a tool for user story quality evaluation: Trustworthy out of the box? In *International Conference on Agile Software Development*, pages 173–181. Springer.

Reine Santos, Gabriel Freitas, Igor Steinmacher, Tayana Conte, Ana Carolina Oran, and Bruno Gadelha. 2025. User stories: Does ChatGPT do it better? In *Proceedings of the 27th International Conference on Enterprise Information Systems*, pages 47–58.

Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, Jacopo Staiano, Patrick Gallinari, and Ionut Sorodoc. 2021. Questeval: Summarization asks for fact-based evaluation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1312–1323, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Shaden Shaar, Nikolay Babulkov, Giovanni Da San Martino, and Preslav Nakov. 2020. That is a known lie: Detecting previously fact-checked claims. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3607–3618.

Amol Sharma and Anil Kumar Tripathi. 2025. Evaluating user story quality with LLMs: A comparative study. *Journal of Intelligent Information Systems*, pages 1–29.

Tjerk Spijkman, Fabiano Dalpiaz, and Sjaak Brinkkemper. 2022. Back to the roots: Linking user stories to requirements elicitation conversations. In *2022 IEEE 30th International Requirements Engineering Conference (RE)*, pages 281–287. IEEE.

Tjerk Spijkman, Xavier de Bondt, Fabiano Dalpiaz, and Sjaak Brinkkemper. 2023. Summarization of elicitation conversations to locate requirements-relevant information. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 122–139. Springer.

Alistair Sutcliffe and Pete Sawyer. 2013. Requirements elicitation: Towards the unknown unknowns. In *Proceedings of the 21st IEEE International Requirements Engineering Conference*, pages 92–104.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: a large-scale dataset for fact extraction and VERification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 809–819, New Orleans, Louisiana. Association for Computational Linguistics.

Stefan Wagner, Daniel Méndez Fernández, Michael Felderer, Antonio Vetrò, Marcos Kalinowski, Roel Wieringa, Dietmar Pfahl, Tayana Conte, Marie-Therese Christiansson, Desmond Greer, Tomi Männistö Casper Lassenius, Maleknaz Nayebi, Markku Oivo, Birgit Penzenstadler, Rafael Prikladnicki, Guenther Ruhe, André Schekelmann, Sagar Sen, Rodrigo Spínola, and 3 others. 2019. Status quo in requirements engineering: A theory and a global family of surveys. *ACM Transactions on Software Engineering and Methodology*, 28(2):1–48.

Alex Wang, Kyunghyun Cho, and Mike Lewis. 2020. Asking and answering questions to evaluate the factual consistency of summaries. In *Proceedings of the*

58th Annual Meeting of the Association for Computational Linguistics, pages 5008–5020, Online. Association for Computational Linguistics.

Asma Yamani, Malak Baslyman, and Moataz Ahmed. 2025. Leveraging LLMs for user stories in AI systems: UStAI dataset. In *Proceedings of the 21st International Conference on Predictive Models and Data Analytics in Software Engineering*, pages 21–30.

Olesya Zaremba and Sotirios Liaskos. 2021. Towards a typology of questions for requirements elicitation interviews. In *Proceedings of the 29th IEEE International Requirements Engineering Conference*, pages 384–389. IEEE.

Yuheng Zha, Yichi Yang, Ruichen Li, and Zhiting Hu. 2023. AlignScore: Evaluating factual consistency with a unified alignment function. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11328–11348, Toronto, Canada. Association for Computational Linguistics.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *Preprint*, arXiv:2306.05685. NeurIPS 2023 Datasets & Benchmarks track.

A Prompts and Templates

This appendix reports the complete prompts used in our experiments. Each prompt was composed of a *system prompt* and a *user prompt*, following standard LLM prompting conventions.

A.1 User Story Generation Prompts

System Prompt

System Prompt

You are an expert requirements engineer and agile coach.
Your task is to read interviews with stakeholders and derive consistent user stories.
– A user story is a **short**, **simple** description of a feature told from the perspective of the person who desires the new capability: a user or customer of the system.
– A user story includes at least a type of user and a goal.
– A user story expresses a requirement for exactly one feature.
– A user story contains nothing more than type of user, goal and reason.

You **MUST NOT** use brackets.
User stories must be written in the standard format:
`"As a <type of user>, I want to <goal>, so that <reason>"`
Strictly follow this format.

User Prompt

User Prompt

Here are some examples of valid user stories:

`As a customer, I want to reset my password online, so that I can regain access without calling support.`
`As a project manager, I want to view a dashboard of team progress, so that I can monitor deadlines.`
`As a student, I want to download lecture slides, so that I can study offline.`

Write a complete set of user stories deduced from the following interview transcript:
`<specification>`

Return all distinct user stories needed to cover the requirements in the transcript, with no overlap. Generate **at most 50** user stories.

Output rules:
Only return the stories separated by newline.
Use the standard format: `"As a <type of user>, I want to <goal>, so that <reason>"`.
One user story per line.
No bullet points, numbering, markdown or commentary — **only** the stories.

A.2 LLM-as-a-Judge Prompts (Matcher X)

System Prompt

System Prompt

You are a senior requirements analyst.
Your task: decide whether the **Chunk Text** gives *enough evidence* to *justify* the **User Story**.

Decision rules
– Output `1` if a knowledgeable reader could infer the User Story from the Chunk Text alone.
– Otherwise output `0`.

Note: Each Chunk Text is an excerpt from an informal interview transcript. Expect colloquial language, filler words (“uh”,

"you know"), and broken grammar; ignore these and focus on meaning.

Additional guidance
– User stories follow the format: `"As a <type of user>, I want to <goal> so that <reason>."`
– Pay special attention to the **goal** in the User Story.
– Verify that the **type of user** (e.g., "employee") is consistent with the Chunk Text.

Example (1 of 4)
`User Story: As a match official, I want to report on match events live, so that I do not register them twice.`
`Chunk Text: <excerpt about referee recording events in real time>`
Answer: `1`
`... (three more few-shot examples omitted for brevity) ...`

Now answer for the next pair.
Return **only one character**, either `1` or `0`, followed by nothing else.

Full prompt available in the project repository at `https://anonymous.4open.science/r/txt2stories-A3FD`.

User Prompt

User Prompt

```
User Story: <story>
Chunk Text: <chunk>
Answer:
```

B Additional Results

B.1 Threshold Calibration in Alignment

Cross-encoders output continuous similarity scores that require a decision threshold to classify pairs. In the main results (Table 3), we reported the *best threshold per dataset*, which constitutes an optimistic upper bound. Here, we examine a more realistic setting in which thresholds are calibrated on one domain and then transferred to another. Specifically, we calibrate thresholds on the two private student projects (Student 1 and Student 2) and evaluate performance on the Public Interview dataset.

Model	Best	Calibrated
Qwen3-Reranker-0.6B	**0.748**	0.722
Qwen3-Reranker-4B	**0.797**	0.750
Qwen3-Reranker-8B	**0.829**	0.744

Table 8: Macro F1-scores on the Public Interview dataset for cross-encoders. Calibration is performed on Student 1 and Student 2.

Mode	Tokens (%)	Time (s)
Direct	100	1,212
Blocked (0.9 recall)	22	329

Table 9: Runtime on Student Project 12. Blocking reduces both tokens and time by around $4\times$.

As shown in Table 8, performance drops when thresholds are calibrated on a different domain. This confirms that cross-encoders are sensitive to threshold selection, limiting their practicality compared to LLM judges that do not require such calibration.

B.2 Blocking impact on runtime

To complement token-based metrics, we benchmark runtime on the dataset with highest number of chunks (Student Project 12). Table 9 shows that blocking reduces runtime by almost a factor of 4. Experiments were run on one NVIDIA A100 (80GB), using Python 3.12 and vLLM 0.10.1.1.

B.3 Blocking efficiency results for a given recall level

At recall 0.9, Figure 6 shows that reductions exceed $4\times$ in some datasets (e.g., Student Projects 3 and 5). Larger interviews yield the strongest gains: Student Projects 12, 5, 3, and 7 (714, 564, 546, and 450 chunks, respectively) require the least percentage of tokens. Fine-tuning adds about 4 points of efficiency on average, and up to 10 points in the best case (Student Project 4).

B.4 Inter-Annotator Agreement

To evaluate the consistency of the annotations, we measured the inter-annotator agreement on a subset of the annotated datasets. The first author initially annotated all chunk–story pairs following the protocol described in Section 5. To evaluate reliability, we randomly selected five user stories from the two student projects used for manual annotation (Student 1 and Student 2). Random sampling was performed using a fixed seed, and the exact selection script is included in the public repository.

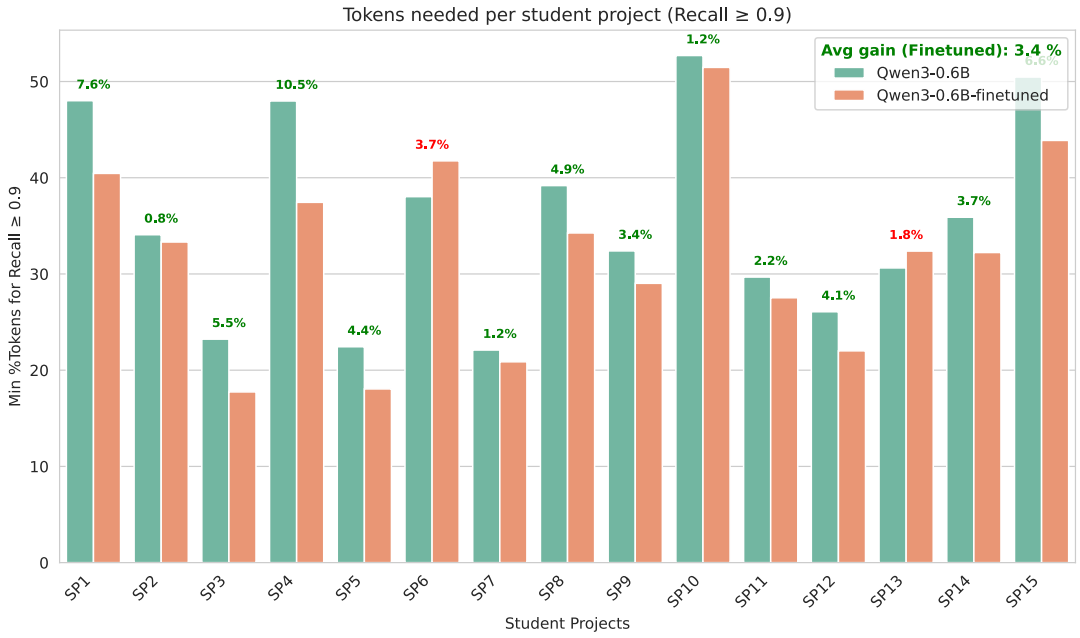

Figure 6: Percentage of tokens that needs to be retrieved by the blocking operator (B_K) per dataset to achieve recall 0.9. Lower values indicate stronger efficiency.

For each of the ten selected user stories (five per project), we extracted the corresponding chunks previously annotated by the first author and submitted them for re-annotation to the two co-authors, who had not participated in the initial dataset annotation. Each annotator independently judged whether each chunk supported the story (1) or not (0).

We then computed Fleiss' κ across the three annotators. The overall agreement is κ=0.470, which corresponds to a *moderate* level of agreement according to standard interpretation scales.

The CSV files containing the full set of survey responses (10 user stories, 5 per project) are available in the GitHub repository.

B.5 Completeness - Human vs. LLM

To better understand why LLM-generated stories are more complete than human-written ones, we conducted a qualitative audit of one representative dataset, Student Project 15, which contains 39 human-written user stories. It is important to note that LLM outputs are capped at 50 stories, while human sets vary in size and scope (although the students were instructed on producing roughly 50 user stories). We manually annotated each story with the functionality it expresses (e.g., "user authentication", "event scheduling", "resource allocation"). This produced a reference list of functionalities covered by the human set. Using the TEXT2STORIES alignment output, we identified the chunks covered by LLM stories but not by any human story (LLM diff Human), as well as the chunks covered by human stories but not by LLM ones (Human diff LLM). Then, we retrieved the stories responsible for covering those exclusive chunks. The result was 19 unique LLM stories in the LLM diff Human set and seven unique human stories in the Human diff LLM set. This suggests that the LLM set's higher completeness is not due to a few stories spanning many chunks, but rather, to the generation of a larger number of distinct functionalities. We manually annotated the functionality expressed by each of the 19 unique LLM stories and compared them with the functionalities in the human set. Seven of these stories corresponded to new functionalities not present in any of the 39 human stories. This is not surprising, as the dataset comes from a student project, where it is reasonable that some functionalities may have been overlooked or incompletely specified. We annotated the closest equivalent human story that expressed the same functionality for the remaining twelve. The analysis confirms that the LLM introduced additional functionalities. The three CSV files containing annotated data from this section can be found in the GitHub repository.