Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

SPERT-AR: A Unified Framework for Resolving User Story Ambiguities to Improve Story Point Estimation

Waleed Younas[1] | **Jing Zhao**[1] | **Tahreem Iqbal**[1] | **Muhammad Ali Lodhi**[2] | **Rui Chen**[1]

[1]School of Software Technology, Dalian University of Technology, Dalian, China

[2]School of Information Engineering, Yangzhou University, Yangzhou, China

Correspondence
Corresponding author Jing Zhao, School of Software Technology, Dalian University of Technology, Dalian, China.
Email: zhaoj9988@dlut.edu.cn

Abstract

Accurate story point estimation remains a core challenge in Agile Software Development because it directly affects sprint planning, resource allocation, and delivery predictability. Learning-based estimators have improved prediction accuracy, but their performance can degrade when user stories contain vague wording, missing conditions, inconsistent phrasing, or overlapping scope. This paper presents SPERT-AR, a framework designed to evaluate whether ambiguity-aware refinement of user stories can improve story point estimation while preserving the original repository-assigned estimation labels. SPERT-AR detects lexical, syntactic, semantic, and pragmatic ambiguity, refines unclear stories using targeted GPT-4 prompts, and applies expert validation to preserve functional intent. The clarified stories are then used as input to a reinforced transformer estimator. To avoid conflating textual clarification with label revision, the primary evaluation keeps the original repository-assigned story points unchanged for both original and resolved stories. Evaluated on ten real-world GitLab projects, SPERT-AR reduces Mean Absolute Error by 29.52% and improves Standardized Accuracy by 23.41% relative to the SPERT baseline under this fixed-label setting. Beyond the headline accuracy results, the findings show that SPERT-AR reduces ambiguity while preserving semantic intent, improves estimation stability across baselines, and provides ablation evidence that ambiguity-aware refinement contributes to the observed improvement beyond model complexity alone. Overall, the findings suggest that clearer user-story representations can make story point estimation more reliable, while expert-adjusted effort interpretations should be treated as secondary evidence rather than primary ground truth.

KEYWORDS
Story Point Estimation, Ambiguity Resolution, Transformer Models, User Stories, Effort Prediction

1 | INTRODUCTION

Accurate story point estimation is a longstanding challenge in Agile Software Development (ASD). Teams rely on stable estimates to plan sprints, balance workloads, and maintain predictability in software delivery Mahmood et al. (2021), Foschini (2021). Despite the widespread use of expert-driven techniques such as planning poker, consensus meetings, and analogy-based reasoning Edison et al. (2021), Yalçıner et al. (2024), Poženel et al. (2023), estimation accuracy often varies substantially across projects and even within the same organization. Uneven team experience, cognitive bias, inconsistent story-writing practices, and continuously evolving requirements all contribute to this variability. The CHAOS Report underscores these concerns, noting that only 36% of projects consistently meet planned time, scope, and budget targets Group (2018).

To address these limitations, researchers have increasingly turned to machine learning (ML) approaches to improve estimation accuracy and reduce dependence on subjective judgment Zhang et al. (2019), Rathore and Kumar (2019), Sánchez-García et al. (2024). Classical ML models and neural architectures, including recurrent networks, transformers, and reinforcement-enhanced predictors, have shown promising gains Malgonde and Chari (2019), Abo-eleneen et al. (2023). However, most of these approaches treat user stories as fixed textual inputs and implicitly assume that they are clear, complete, and detailed enough for models to extract meaningful effort cues. In practice, this assumption often fails because user stories may contain vague wording, missing conditions, implicit assumptions, or overlapping functionality Tawosi et al. (2022a). Such ambiguity is not the only cause of estimation error, but it introduces linguistic noise that can disrupt the patterns learning-based

estimators depend on, leading to inconsistent or biased predictions when models are trained on unrefined stories Tawosi et al. (2022b).

Existing story point estimation models mainly improve the prediction algorithm while paying limited attention to the clarity of the input requirements. Agile user stories may contain lexical vagueness, syntactic incompleteness, semantic underspecification, or pragmatic overlap with other stories. These ambiguities can obscure effort-related cues and make it difficult to determine whether estimation errors arise from model limitations or unclear input requirements.

This study considers ambiguity across four dimensions: lexical ambiguity caused by vague or polysemous terms, syntactic ambiguity caused by unclear sentence structure, semantic ambiguity caused by missing context or assumptions, and pragmatic ambiguity caused by overlap or inconsistency across related stories. In the collected GitLab corpus, 3,559 out of 4,252 stories contain at least one detected ambiguity, indicating that ambiguity is a frequent issue in the studied repositories rather than an isolated textual defect.

SPERT-AR is designed to: (i) identify ambiguity in user-story text, (ii) refine unclear stories while preserving functional intent, (iii) validate refinements through expert review, and (iv) evaluate whether clarified textual representations improve story point estimation under unchanged repository-assigned labels.

Existing estimation techniques generally treat ambiguity as a minor pre-processing step, if they acknowledge it at all. Meanwhile, research on ambiguity detection and rewriting has evolved largely independently, focusing on requirements quality, defect prevention, or stakeholder communication Fichtinger and Lübke (2015), Faris et al. (2021). Consequently, the effect of explicit ambiguity reduction on downstream story point estimation remains underexplored. To the best of our knowledge, no prior work integrates ambiguity modeling, LLM-based refinement, expert validation, and reinforcement-driven estimation within a unified framework for story point estimation.

This paper addresses this gap by extending our earlier Story Point Estimation using Reinforced Transformers (SPERT) model Younas et al. (2025) and introducing SPERT-AR. The framework operates in three coordinated stages: it identifies ambiguity across lexical, syntactic, semantic, and pragmatic dimensions; resolves unclear stories using structured GPT-4 prompts OpenAI (2023) followed by expert validation; and trains a reinforced transformer estimator on the clarified textual representations. By linking input clarity, linguistic refinement, and adaptive prediction, SPERT-AR aims to improve the reliability of story point estimation in practical Agile settings.

We evaluate SPERT-AR on ten publicly available GitLab projects comprising 4,252 user stories. Under a fixed-label evaluation protocol that preserves the original repository-assigned story points, SPERT-AR reduces Mean Absolute Error (MAE) by 29.52% and substantially improves cross-project stability compared with the SPERT baseline (see Section 5). These results indicate that ambiguity-aware textual refinement can improve estimation performance without redefining the original ground-truth labels.

The main contributions of this work are as follows:

- We introduce SPERT-AR, a unified framework that integrates ambiguity detection, targeted ambiguity resolution, expert validation, and reinforcement-based estimation within a single workflow.
- We develop a structured multi-dimensional ambiguity model covering lexical, syntactic, semantic, and pragmatic uncertainty, together with GPT-4 prompt templates designed to refine unclear user stories while preserving their functional intent.
- We construct a dual-view dataset of 4,252 user stories from ten GitLab projects, including original stories, ambiguity annotations, and expert-validated clarified versions. The original repository-assigned story points are retained as the primary ground-truth labels for the main experiments, while expert-suggested effort reinterpretations are recorded only as secondary exploratory annotations.
- We provide empirical evidence that ambiguity-aware textual refinement improves prediction accuracy, cross-project generalization, and estimator stability across multiple baselines under a fixed-label evaluation protocol that keeps the original repository-assigned story points unchanged.

To guide the empirical evaluation, Table 1 summarizes the research questions and their corresponding objectives.

TABLE 1 Research questions and objectives of the study.

RQ	Objective
RQ1	Assess whether ambiguity resolution improves story point estimation when original and resolved stories are evaluated using the same repository-assigned story-point labels.
RQ2	Compare SPERT-AR with traditional and learning-based baselines, including TF-IDF-SE, Deep-SE, SBERT-XG, and SPERT.
RQ3	Evaluate whether SPERT-AR maintains stable performance under cross-project evaluation on previously unseen repositories.
RQ4	Analyze the computational and operational overhead introduced by ambiguity resolution, LLM-based refinement, expert validation, and reinforcement learning.

The remainder of the paper is organized as follows: Section 2 reviews existing work on effort estimation and ambiguity handling. Section 3 describes the SPERT-AR framework. Section 4 presents the dataset and annotation methodology. Section 5 details the experimental setup and results. Section 6 reflects on practical implications, Section 7 discusses threats to validity, and Section 8 concludes the study.

2 | RELATED WORK

Research on story point estimation has evolved along two largely independent lines: (i) data-driven and learning-based models that attempt to improve prediction accuracy Kapur and Sodhi (2022), Mahmood et al. (2021), Choetkiertikul et al. (2018), and (ii) techniques for detecting

and resolving ambiguity in natural-language requirements Amna and Poels (2022), Moharil and Sharma (2023). Although both streams have advanced significantly, they rarely intersect. Existing estimators generally assume that user stories are clear and complete, while ambiguity research seldom evaluates downstream effects on effort prediction. Accordingly, we review prior work by examining how existing methods represent user-story text for estimation, how ambiguity-detection and refinement methods improve requirements quality, and whether clarified stories are evaluated for their effect on story point estimation.

2.1 | Story Point Estimation

Traditional software cost estimation techniques, such as COCOMO Rashid et al. (2025) and Function Point Analysis Putri and Subriadi (2019), were designed for documentation-heavy development processes. These methods assume stable, formally expressed requirements, which makes them difficult to apply in Agile environments where requirements evolve continuously and user stories often lack detail Gurung et al. (2020), Pargaonkar (2023). As Agile practices matured, researchers explored data-driven alternatives to increase consistency and reduce reliance on expert judgment. Early machine learning (ML) approaches used decision trees, support vector machines, and linear regressors trained on handcrafted textual or numeric features Ferrucci et al. (2019), Tran et al. (2023). While these models improved upon informal estimation strategies, they remained sensitive to project-specific vocabulary and often struggled to generalize across teams and domains.

The growing use of ML and deep learning in story point estimation reflects a broader trend in software engineering and software-intensive systems, where predictive models support decision-making under noisy and heterogeneous inputs. In software defect prediction, ensemble learning and feature selection have been shown to improve prediction quality when software metrics or textual features vary across contexts Ali et al. (2023 2024). Similar AI-based methods have also been applied to IoT security, wireless sensor network security, and renewable-energy forecasting, demonstrating the broader value of learning-based prediction and classification models Mazhar et al. (2023), Ghadi et al. (2024), Khan et al. (2024). In Agile estimation, recent work further shows that story-based effort estimation is shaped by story quality, team communication, and task complexity Iqbal et al. (2024). These findings reinforce the need to examine not only the estimator but also the clarity and structure of the user-story text used as input.

Deep learning approaches later addressed some limitations of classical ML. RNN, LSTM, and CNN-based models demonstrated stronger ability to capture contextual cues within user stories Choetkiertikul et al. (2018). Transformer-based representations further improved robustness by leveraging contextual embeddings from large pre-trained language models such as BERT and SBERT Reimers and Gurevych (2019), Fu and Tantithamthavorn (2022). More recently, reinforcement learning introduced feedback-driven optimization: SPERT Younas et al. (2025) demonstrated that a reinforced transformer could penalize both over- and underestimation to learn more balanced prediction policies. However, these approaches still largely treat user stories as fixed textual artifacts. In practice, industrial user stories frequently contain vague terms, missing conditions, and inconsistent phrasing Butt et al. (2023), which can weaken the effort signals that learning-based estimators rely on.

Taken together, prior estimation studies show clear progress from handcrafted features to contextual embeddings and reinforced predictors. Their main limitation, however, is that they focus primarily on improving the prediction model while rarely questioning whether ambiguous input text reduces the reliability of the learned mapping from stories to story points. This limitation motivates SPERT-AR's focus on ambiguity-aware refinement as a preceding step to estimation.

2.2 | Ambiguity in User Stories

Ambiguity in natural-language requirements is widely recognized as a source of misunderstanding, rework, and inaccurate effort estimation. Prior studies report that ambiguous or incomplete requirements increase project risk and planning uncertainty Amna and Poels (2022), Venkataraman and Pinto (2023). For example, a story such as *"As a user, I want the system to respond quickly"* contains lexical ambiguity because the term "quickly" lacks a measurable target, while *"As an admin, I want to manage users and generate reports"* combines multiple tasks and may lead to inconsistent estimates. Empirical analysis by Amna and Poels shows that many real-world user stories contain at least one form of linguistic ambiguity. Such ambiguity commonly appears across lexical, syntactic, semantic, and pragmatic levels, each affecting how developers interpret functional intent Fichtinger and Lübke (2015), da Silva Neo et al. (2024).

Several techniques have been proposed to mitigate these issues. Traditional approaches rely on structured checklists, peer review, or controlled natural languages to improve requirement clarity Fabbrini et al. (2001). Although useful, these practices require substantial manual effort and are difficult to maintain in fast-paced Agile environments. Early automated techniques used pattern matching, rule-based heuristics, or restricted vocabularies to detect ambiguity Osama and Aref (2018), Dar et al. (2024). These methods are effective in controlled settings but often struggle with informal and project-specific user-story phrasing. Ontology-based methods improve semantic consistency checking Antoniou and Bassiliades (2024), but they typically depend on manually curated knowledge bases.

More recently, large pre-trained language models have improved ambiguity detection and refinement. Models such as BERT and RoBERTa support ambiguity classification by capturing contextual representations of requirement text Devlin et al. (2019), Moharil and Sharma (2023). Generative models based on GPT further enable automated rewriting and clarification of requirements OpenAI (2023). However, automated rewriting can unintentionally modify scope or introduce functionality that was not present in the original story. For instance, rewriting *"As a user, I want to register"* may add assumptions such as email verification or role assignment. Therefore, expert oversight remains necessary when applying LLM-based rewriting to requirements.

SPERT-AR differs from generic LLM-based refinement approaches in both purpose and evaluation. In LLM-only refinement, the clarified requirement is often treated as the final output. In contrast, SPERT-AR uses LLM-generated clarification as an intermediate controlled transformation whose value is assessed through downstream story point estimation. The framework constrains rewriting through ambiguity-type prompts, semantic-retention checks, and expert validation, thereby reducing the risk of scope drift. Its contribution is therefore not simply the use of GPT-4 for rewriting, but the integration of ambiguity-aware refinement with a fixed-label estimation protocol.

Complementary work has proposed structured frameworks for representing ambiguity. Alomari et al. Alomari and Elazhary (2018) introduced finite-state templates to constrain syntactic variation, while Ashfaq et al. Ashfaq and Bajwa (2021) applied SBVR-based representations to model requirement semantics more precisely. Rahmawati et al. Rahmawati et al. (2025) later proposed AmbiTRUS, a multi-dimensional framework for quantifying ambiguity in user stories. Although these frameworks provide systematic mechanisms for detecting or representing ambiguity, they are rarely integrated with downstream effort-estimation tasks.

In summary, story point estimation research has advanced from classical ML models to neural and reinforced transformer-based estimators, while ambiguity research has developed increasingly capable detection, classification, and refinement techniques. The connection between these areas remains limited: estimation models usually improve the predictor while treating the story as a fixed input, whereas ambiguity-focused methods improve requirement quality without testing whether clarification improves downstream estimation. SPERT-AR addresses this gap by combining ambiguity detection, controlled LLM-based refinement, expert validation, and fixed-label story point estimation in one evaluation framework.

3 | SPERT-AR FRAMEWORK

SPERT-AR extends the reinforced transformer backbone of SPERT by explicitly modeling and resolving linguistic ambiguity in Agile user stories before effort estimation. The framework treats ambiguity as a measurable property of text, reduces it through a combination of LLM-based rewriting and expert validation, and then learns an adaptive prediction policy on the clarified stories. The overall goal is to link input clarity, formal ambiguity modeling, and reinforcement learning within a single, practically deployable workflow.

At a high level (Figure 1), SPERT-AR operates in three stages. First, it detects and categorizes ambiguity in user stories using a formal multi-dimensional ambiguity model combined with GPT-4 prompts and embedding-based heuristics. Second, it refines ambiguous stories through targeted rewriting and expert validation, producing a parallel corpus of original and resolved user stories. Third, it performs story point estimation on the resolved stories using a transformer encoder trained with a hybrid supervised and reinforcement learning objective. This section details each component and the associated training strategy.

From a deployment perspective, SPERT-AR is designed as a modular pipeline that can be connected to existing backlog-management platforms rather than as a standalone replacement for Agile estimation practices. The ambiguity-identification module can be triggered when a user story is created or updated in tools such as GitLab, Jira, or Azure DevOps. Stories with low ambiguity can proceed directly to estimation, whereas stories exceeding the ambiguity threshold can be routed to the resolution module and then to human validation. This modular design allows teams to adopt SPERT-AR incrementally: ambiguity detection can be used first as a quality-checking layer, LLM-based refinement can be added for high-risk stories, and reinforced estimation can be used once a sufficient clarified-story corpus is available.

Table 2 summarizes the main workflow components of SPERT-AR, including the input, function, and output of each module.

3.1 | Formal Ambiguity Model

Let a user story U_i be a text pair (T_i, D_i), where T_i is the title and D_i is the description. SPERT-AR concatenates them into a single sequence following the BERT convention:

$$U_i = [\text{CLS}]\, T_i\, [\text{SEP}]\, D_i\, [\text{SEP}]. \tag{1}$$

Ambiguity is modeled along four orthogonal dimensions

$$\mathcal{T} = \{\text{lexical}, \text{syntactic}, \text{semantic}, \text{pragmatic}\}, \tag{2}$$

consistent with established taxonomies in requirements engineering. For each type $k \in \mathcal{T}$, a non-negative function $\phi_k(U_i) \geq 0$ measures the intensity of that ambiguity in U_i. The overall ambiguity score is defined as a weighted combination:

$$A(U_i) = \sum_{k \in \mathcal{T}} w_k\, \phi_k(U_i), \qquad w_k \geq 0,\ \sum_{k \in \mathcal{T}} w_k = 1. \tag{3}$$

Ambiguity resolution is formulated as a constrained optimization problem. Given the space $\mathcal{U}$ of possible rewrites of U_i, SPERT-AR seeks a refined story $\hat{U}_i$ that minimizes the ambiguity score while preserving the original semantics:

$$\hat{U}_i = \arg\min_{U_i' \in \mathcal{U}} A(U_i') \quad \text{s.t.} \quad \text{Sim}(U_i, U_i') \geq \delta, \tag{4}$$

where $\text{Sim}(\cdot,\cdot)$ denotes cosine similarity between sentence embeddings and δ is a semantic similarity threshold. In our experiments, we set $\delta = 0.85$ based on validation performance. The operator $\mathcal{R}$ induced by (4) defines the mapping

$$\hat{U}_i = \mathcal{R}(U_i), \tag{5}$$

yielding a disambiguated version of the story that preserves functional meaning and development scope. Expert validation of $\hat{U}_i$ then produces

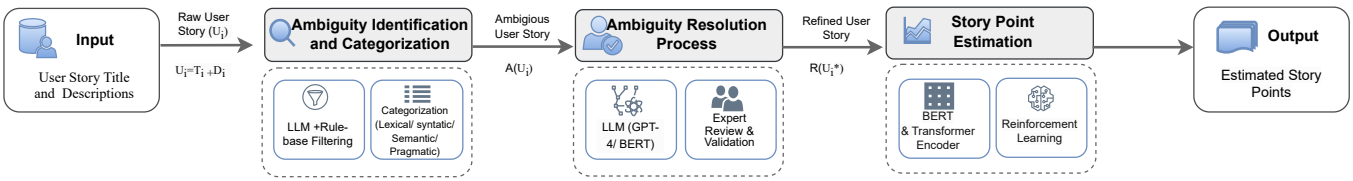

FIGURE 1 Overview of the SPERT-AR framework.

TABLE 2 Workflow components of the SPERT-AR framework.

Component	Input	Main function	Output
Story acquisition	User story title and description	Constructs the textual input sequence for ambiguity analysis and estimation.	Normalized user-story representation
Ambiguity detection	Original user-story text	Identifies lexical, syntactic, semantic, and pragmatic ambiguity using GPT-4 prompts, semantic coherence checks, and project-level similarity analysis.	Ambiguity labels and confidence scores
Ambiguity resolution	Original story and detected ambiguity labels	Generates targeted clarifications using ambiguity-type-specific GPT-4 prompts while preserving functional intent.	Candidate resolved story
Expert validation	Original and candidate resolved stories	Checks semantic fidelity, completeness, and scope preservation before accepting or editing the refinement.	Expert-validated resolved story
Feature representation	Validated resolved story	Encodes the story using transformer-based textual representations.	Contextual embedding vector
Story point estimation	Contextual embedding and repository-assigned label during training	Trains the reinforced transformer estimator and predicts story points for clarified stories.	Predicted story point

a corrected and confirmed version U_i^* and a validation decision indicating whether the refinement is accepted, edited, or rejected before inclusion in the resolved textual corpus.

3.2 | Ambiguity Identification and Categorization

The first operational stage in SPERT-AR is ambiguity identification and categorization (Figure 2). Its role is to locate, type, and quantify uncertainty in each story before any rewriting occurs.

Ambiguity detection is modeled as a multi-label classification task with a binary vector $\mathbf{a}_i \in \{0,1\}^4$ representing the four ambiguity categories:

$$\mathbf{a}_i = f_{\text{detect}}(U_i), \tag{6}$$

where f_{detect} combines GPT-4 prompt-based classification with statistical and embedding-level heuristics.

For each ambiguity type $k \in \mathcal{T}$, a structured GPT-4 prompt produces a confidence score indicating the presence of that type. The probabilistic output is modeled as:

$$\mathbf{a}_i^{(k)} = \sigma\big(f_{\text{GPT-4}}(U_i, k)\big), \tag{7}$$

where σ is the sigmoid function. Predictions above a threshold τ are marked as positive for type k. Prompt templates are specialized for each dimension: lexical prompts target vague quantifiers and polysemous terms; syntactic prompts assess sentence completeness and clause structure; semantic prompts evaluate information sufficiency; and pragmatic prompts look for potential overlap with neighboring stories in the same project. This hybrid use of LLM reasoning and linguistic cues enables the detection of ambiguity patterns that simple surface features would miss.

To complement GPT-4 predictions, SPERT-AR computes a semantic coherence score that measures internal consistency within each story. Key phrases $\mathcal{P}_i$ are extracted using part-of-speech patterns and ranked by TF–IDF Wu et al. (2008). For a sampled set of phrase pairs $\mathcal{Q}_i \subseteq \mathcal{P}_i \times \mathcal{P}_i$, cosine similarity between SBERT embeddings is averaged as:

$$\alpha_i = \frac{1}{|\mathcal{Q}_i|} \sum_{(p,q)\in\mathcal{Q}_i} \cos\big(\text{SBERT}(p), \text{SBERT}(q)\big). \tag{8}$$

Low values of α_i (below a threshold λ) indicate insufficient semantic coherence and suggest missing or implicit contextual elements.

At the project level, pragmatic ambiguity is inferred from overlap between user stories. A similarity matrix $S \in \mathbb{R}^{n \times n}$ is constructed as:

$$S_{ij} = \cos\left(\text{SBERT}(U_i), \text{SBERT}(U_j)\right), \tag{9}$$

where n is the number of stories in the project. If $S_{ij} > \gamma$ (typically $\gamma = 0.85$), the pair (U_i, U_j) is flagged as sharing functional intent or scope and marked for potential pragmatic conflict. This mechanism highlights redundant or duplicated requirements that can distort perceived effort.

The final ambiguity label $\mathbf{a}_i$ consolidates GPT-4 outputs, semantic coherence scores, and pragmatic similarity measures. A type is labeled positive if detected by any component. This hybrid integration improves recall while maintaining acceptable precision, providing reliable signals for the subsequent resolution step.

3.2.1 | Detection Thresholds and Reproducibility Settings

To improve reproducibility, the ambiguity identification module uses fixed thresholds and deterministic prompting settings across all projects. For each ambiguity type $k \in \mathcal{T}$, the GPT-4 classifier returns a confidence score $p_i^{(k)} \in [0, 1]$. A story is marked as positive for ambiguity type k when $p_i^{(k)} \geq \tau$, where $\tau = 0.70$ in all experiments. The semantic coherence threshold is set to $\lambda = 0.60$; stories with average phrase-level coherence $\alpha_i < \lambda$ are flagged for semantic insufficiency. Pragmatic overlap is detected when the project-level story-pair similarity satisfies $S_{ij} > \gamma$, with $\gamma = 0.85$. For the resolution stage, the semantic-retention constraint in Eq. 4 uses $\delta = 0.85$, meaning that a GPT-4 refinement is accepted only if its SBERT cosine similarity with the original story is at least 0.85 before expert review.

All GPT-4 calls are executed with temperature 0.0 to reduce response variability. The maximum output length is limited to 512 tokens for ambiguity classification and 768 tokens for ambiguity resolution. For each story, one classification response and one resolution response are generated per active ambiguity type. The prompts instruct the model to preserve the original functional intent, avoid adding new features, avoid changing acceptance criteria beyond clarification, and explicitly state the detected ambiguity type and the proposed refinement. These settings ensure that the ambiguity-resolution process behaves as a controlled textual refinement step rather than an unconstrained story-generation process.

3.2.2 | Prompt Template Structure

The ambiguity detection prompt follows a fixed structure: (i) the original user story title and description, (ii) the ambiguity type to be checked, (iii) a short operational definition of that ambiguity type, and (iv) an instruction to return a binary label, confidence score, and brief justification. The resolution prompt uses the detected ambiguity labels and asks GPT-4 to produce a clarified version under four constraints: preserve the original intent, avoid scope expansion, make implicit conditions explicit only when supported by the original text, and keep the revised story suitable for Agile story-point estimation. The same prompt templates are applied across all ten projects, with only the story text and detected ambiguity type changing between calls.

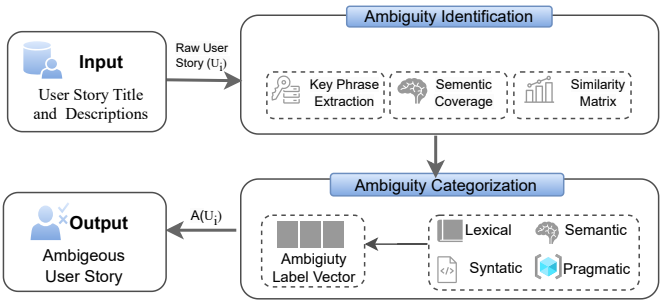

FIGURE 2 Ambiguity identification and categorization pipeline in SPERT-AR.

3.3 | Ambiguity Resolution

Once ambiguity types are identified, SPERT-AR refines user stories through a targeted rewriting mechanism (Figure 3). The objective is to transform each ambiguous story into a syntactically valid, semantically complete, and pragmatically scoped version suitable for downstream learning.

Formally, the transformation is modeled as:

$$\tilde{U}_i = f_{\text{resolve}}(U_i, \mathbf{a}_i), \tag{10}$$

where f_{resolve} is a conditional rewriting operator driven by GPT-4 and parameterized by the detected ambiguity types. Each active type in $\mathbf{a}_i$ activates a corresponding prompt template $P_a(U_i)$ from a predefined library $\mathcal{P}$. Lexical templates instruct GPT-4 to replace vague quantifiers with measurable targets, syntactic templates enforce grammatical completeness or split compound stories, semantic templates request missing preconditions and constraints, and pragmatic templates remove overlapping functionality or consolidate redundant stories.

The composed prompt aggregates all applicable templates:

$$P(U_i) = \bigcup_{a \in \mathcal{A}_i} P_a(U_i), \tag{11}$$

where $\mathcal{A}_i$ is the active set of ambiguity types for U_i. GPT-4 then generates a refined story:

$$\tilde{U}_i = f_{\text{GPT-4}}(U_i, P(U_i)). \tag{12}$$

Each generated refinement is reviewed by Agile practitioners to maintain fidelity and scope alignment. Reviewers compare the original story U_i, its detected ambiguity types $\mathbf{a}_i$, and the refined version $\tilde{U}_i$. Based on

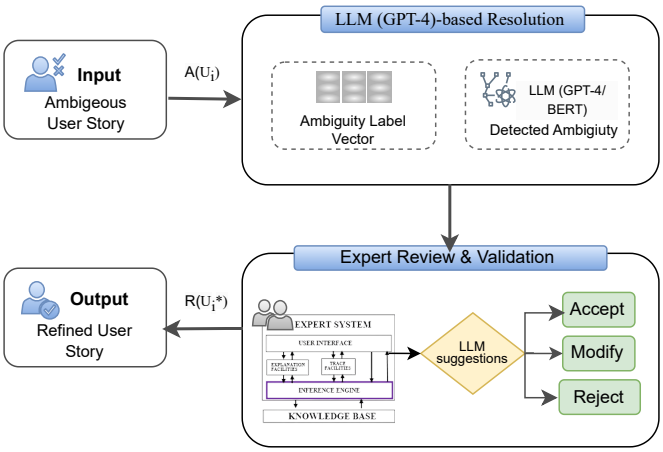

FIGURE 3 Ambiguity resolution workflow in SPERT-AR, combining GPT-4–based rewriting with Agile expert validation.

this comparison, the final resolved story U_i^* is determined as:

$$U_i^* = \begin{cases} \tilde{U}_i, & \text{if accepted,} \\ \tilde{U}_i', & \text{if accepted with edits,} \\ U_{\text{manual},i}, & \text{if rewritten manually.} \end{cases} \tag{13}$$

To prevent scope drift, SPERT-AR also monitors whether a refinement could plausibly change practitioner interpretation of effort. Such cases are flagged for additional expert review and recorded as exploratory effort-interpretation annotations. However, these annotations are stored separately and are not used as the target labels in the main predictive experiments. In our dataset, 53% of GPT-4 rewrites were accepted directly, 34% required minor edits, and 13% were rewritten manually, indicating that LLM-based rewriting is useful but still benefits from human oversight. Expert feedback from this process is used to determine whether each GPT-4 refinement is accepted, edited, or rejected before inclusion in the resolved textual corpus. This feedback serves as a quality-control mechanism for preserving semantic fidelity and preventing scope drift; it is not used as a separate numerical reward or as a replacement for the original repository-assigned story-point label in the main training objective.

The resolution process produces a high-quality parallel dataset of (U_i, U_i^*) pairs, linking ambiguous and resolved versions of each story. This dual-view corpus supports comparative evaluation, ablation studies, and transfer learning, where models trained on ambiguous data can be fine-tuned on resolved data to assess gains in estimation stability.

3.4 | Story Point Estimation with SPERT

The final stage of SPERT-AR estimates story points from the resolved and validated user stories U_i^* using a reinforced transformer module (Figure 4). This module reuses the SPERT architecture, which couples transformer-based representation learning with reinforcement learning for policy adaptation.

Each resolved story U_i^* is encoded in the same BERT-style format:

$$U_i^* = [\text{CLS}]\, T_i\, [\text{SEP}]\, D_i\, [\text{SEP}], \tag{14}$$

and passed through a pretrained BERT model to obtain contextual embeddings:

$$E_i = \text{BERT}(U_i^*). \tag{15}$$

These embeddings are then processed by a multi-layer transformer encoder to derive higher-level representations:

$$H_i = \text{TransformerEncoder}(E_i), \tag{16}$$

which capture both local and long-range dependencies and approximate the underlying story complexity.

A regression head maps H_i to a predicted story point $\hat{y}_i$. Rather than relying solely on supervised regression, SPERT-AR frames estimation as a reinforcement learning problem in which predictions are iteratively adjusted through experience-driven policy updates. The reward for each prediction is defined as:

$$R_i = -|\hat{y}_i - y_i|, \tag{17}$$

which symmetrically penalizes overestimation and underestimation, reflecting Agile practice where both outcomes degrade planning quality.

Policy optimization uses Proximal Policy Optimization (PPO) Schulman et al. (2017), which balances exploration and update stability. The PPO objective is:

$$L^{\text{PPO}} = \mathbb{E}_t[\min(r_t(\theta)A_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)]\,, \tag{18}$$

where $r_t(\theta)$ is the policy ratio, ϵ is the clipping threshold, and A_t is the advantage estimator defined as $A_t = R_t - V_t$ with V_t denoting the value estimate. The actor–critic mechanism Fujimoto et al. (2018) stabilizes learning and improves convergence, particularly in the presence of high variance across projects.

The final prediction is expressed as:

$$\hat{y}_i = \text{ActorCritic}(H_i), \tag{19}$$

where the actor outputs the story point and the critic estimates its expected quality. By combining refined linguistic input with adaptive estimation, SPERT-AR forms a closed learning loop: ambiguity resolution reduces input noise, reinforcement learning optimizes sensitivity to effort signals, and expert validation provides grounded human feedback.

3.5 | Model Training

The SPERT-AR training strategy integrates ambiguity resolution and reinforced estimation into a two-stage process. All models, including baselines (TF-IDF-SE Tawosi et al. (2022b), Deep-SE Choetkiertikul et al. (2018), and SBERT-XG Yalçıner et al. (2024)), are trained on identical train/validation/test splits of the ambiguous and resolved datasets to

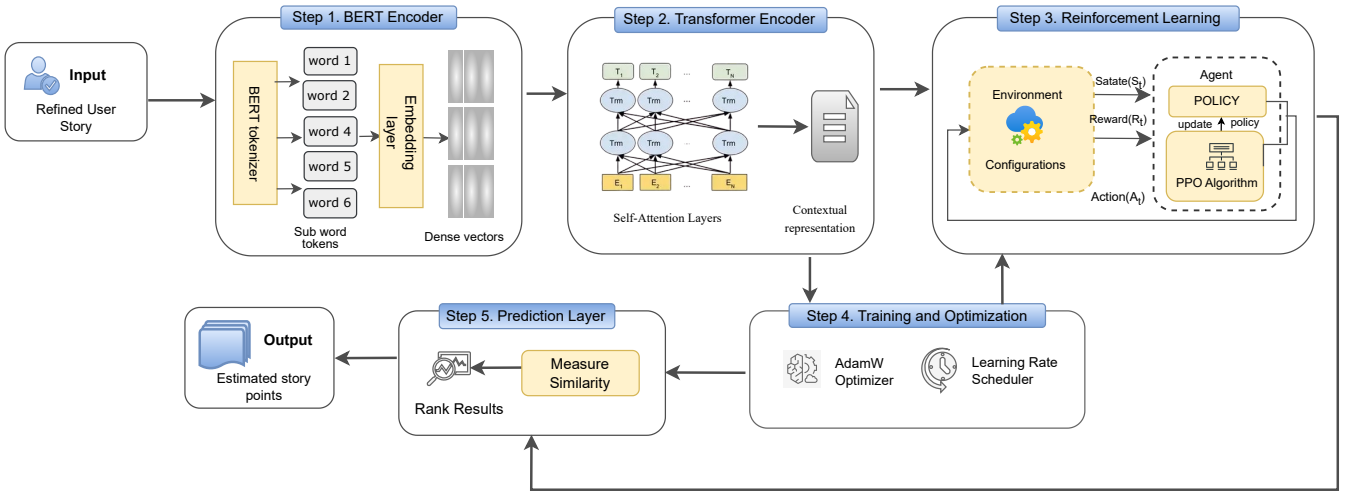

FIGURE 4 SPERT module embedded within SPERT-AR.

ensure fair comparison; SPERT-AR, however, uniquely exploits both ambiguity-aware pre-processing and reinforcement learning.

Stage 1: Ambiguity Resolution and Corpus Construction

In the first stage, GPT-4 is used in inference mode to evaluate each story U_i for the four ambiguity types, producing the label vector $\mathbf{a}_i$. For each active label, a corresponding rewriting prompt is generated, and GPT-4 produces a refined candidate $\tilde{U}_i$ according to the process described in Section 3.3. Human experts validate each candidate for correctness, completeness, and scope alignment; only expert-approved or manually corrected stories are retained as U_i^*. This yields the resolved dataset $\mathcal{D}_{\text{resolved}} = \{(U_i^*, y_i)\}$, where y_i denotes the original repository-assigned story point preserved for primary model training and evaluation. Expert feedback is used to accept, edit, or reject the textual refinement, but it does not redefine the main prediction target. In early iterations, more than 60% of GPT-4 outputs required expert revision. Through iterative prompt tuning and calibration, this proportion was reduced to below 30%, increasing throughput while preserving semantic fidelity. This stage is executed offline and can be reused whenever new stories are added to the backlog.

Stage 2: Hybrid Supervised and Reinforcement Learning

In the second stage, SPERT-AR is trained on the resolved stories U_i^*. Each story is embedded to produce H_i, and a regression head predicts $\hat{y}_i$. Training starts with standard supervised learning to minimize Mean Absolute Error between predicted and true story points, providing a stable initialization for the policy. After this warm-up phase, reinforcement learning is introduced using the reward definition in (17). Policy-gradient updates follow:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[R_i \cdot \nabla_\theta \log \pi_\theta(\hat{y}_i \mid H_i)\right], \tag{20}$$

where π_θ denotes the current policy. In practice, we optimize the clipped PPO objective to stabilize training.

The final hybrid loss combines supervised prediction error with the reinforcement-learning objective while preserving the original repository-assigned labels as the training target:

$$\mathcal{L}_{\text{total}} = \text{MAE}(y_i, \hat{y}_i) \; - \; \beta\, \mathbb{E}_{\pi_\theta}\left[R_i\right], \tag{21}$$

where β controls the weight of the reinforcement term. The supervised component minimizes the error between the prediction $\hat{y}_i$ and the original repository-assigned story point y_i, while the reinforcement component encourages prediction policies that reduce absolute estimation error. Expert validation affects the quality of the input text by determining whether a GPT-4 refinement is accepted, edited, or rejected; it is not used as a separate numerical reward in the main training objective. This separation ensures that the reported improvements are attributable to ambiguity-aware textual refinement and model learning rather than to changes in the target labels. Training is performed for a fixed number of epochs (typically around 50) with early stopping based on validation MAE. Practical challenges such as label imbalance (most stories lie in the 1–3 point range), residual semantic noise, and early-stage RL instability are mitigated through stratified sampling, dropout, gradient clipping, and moving-average baselines. Mixed-precision training, caching, and check pointing are used to maintain reasonable runtime. The empirical impact of this training strategy on convergence speed, accuracy, and computational cost is analyzed in Section 5.

3.5.1 | Model Configuration and Training Settings

For reproducibility, all neural models are trained under fixed configuration settings. The SPERT and SPERT-AR estimators use a BERT-base encoder with 12 transformer layers, 768 hidden dimensions, 12 attention heads, and a maximum input length of 256 tokens. User story titles and descriptions are concatenated using the standard `[CLS]` title `[SEP]` description `[SEP]` format. The transformer estimation head contains two fully connected layers with ReLU activation and dropout of 0.20 before the final regression output.

Training begins with supervised warm-up for 10 epochs using the AdamW optimizer, a learning rate of 2×10^{-5} for the encoder, and 1×10^{-4} for the regression and policy heads. After warm-up, PPO-based policy optimization is applied for up to 40 additional epochs with clipping parameter $\epsilon = 0.20$, discount factor $\gamma_{\mathrm{RL}} = 0.99$, and reinforcement weight $\beta = 0.30$. Mini-batch size is set to 16, gradients are clipped at 1.0, and early stopping is triggered if validation MAE does not improve for five consecutive epochs. All experiments are repeated with five random seeds, and the reported values are averaged across runs.

Baseline models are trained using the same train, validation, and test partitions as SPERT-AR. TF-IDF-SE uses unigram and bigram TF–IDF features with a maximum vocabulary size of 10,000 and a linear regression estimator. Deep-SE uses averaged word embeddings followed by a multilayer perceptron with two hidden layers. SBERT-XG uses sentence embeddings extracted from Sentence-BERT and an XGBoost regressor tuned on the validation set. This common partitioning and validation protocol ensures that differences in performance are attributable to the ambiguity-resolution and estimation mechanisms rather than to inconsistent data splits or tuning procedures.

3.5.2 | Implementation Details

The SPERT-AR implementation is structured as a research prototype with modular components for ambiguity detection, ambiguity resolution, expert validation, feature extraction, model training, and evaluation. The learning components are implemented in Python using transformer-based text encoders and reinforcement-learning routines, while baseline models are trained using their corresponding textual representations and validation settings. GPT-4 is accessed only during the ambiguity detection and resolution stages, and resolved stories are stored so that unchanged backlog items do not require repeated LLM calls.

The prototype does not require a dedicated end-user interface for the experiments reported in this paper; however, its outputs are designed to support integration with backlog-management tools. For each processed story, the system records the original text, detected ambiguity type, confidence score, candidate refinement, expert-validation decision, resolved text, and predicted story point. These outputs can be exposed through a lightweight dashboard, a GitLab/Jira plugin, or an external reporting service in practical deployment.

To improve training efficiency, the implementation uses cached textual representations where possible, early stopping based on validation MAE, gradient clipping during reinforcement learning, and repeated evaluation over multiple random seeds. These design choices reduce unnecessary recomputation, stabilize model training, and support reproducible comparison with baseline estimators.

4 | DATASET AND ANNOTATION

This section describes the dataset used to evaluate SPERT-AR and provides a transparent account of project selection, ambiguity annotation, expert validation, and the resulting corpus characteristics Tawosi et al. (2022a). The dataset comprises user stories sourced from ten Agile projects on GitLab, each paired with ambiguity labels and expert-validated refined versions. The original repository-assigned story points are retained as the primary ground-truth labels for all main experiments, while expert-suggested adjusted story points are recorded only as secondary exploratory annotations.

4.1 | Project Selection

Candidate repositories were first identified from the public GitLab ecosystem.[†] From an initial pool of thirty projects, ten were selected using two criteria: (i) each project contained at least 250 user stories with assigned story points, and (ii) the stories exhibited sufficient linguistic diversity and observable ambiguity to support a meaningful evaluation. Repositories with inconsistent labeling practices, incomplete descriptions, or insufficient story volume were excluded to maintain dataset quality.

Each selected project contributes three aligned artifacts: the original user story (title, description, and assigned story point), the ambiguity annotations across four linguistic dimensions, and the refined version produced through the ambiguity-resolution pipeline. This structure enables parallel comparison between ambiguous and clarified stories and supports cross-project generalization studies. The resulting corpus spans analyzers, security tools, marketing platforms, and static-analysis utilities, providing substantial heterogeneity in writing styles and Agile conventions. Table 3 summarizes story counts and ambiguity prevalence. Notably, ambiguity rates range from 12.6% in the Checkov Analyzer to near complete coverage in several Gemnasium projects, offering a diverse empirical basis for evaluating SPERT-AR across different ambiguity profiles.

The dataset is intentionally ambiguity-rich because the objective of this study is to evaluate whether resolving ambiguity in user stories improves downstream story point estimation. Overall, 3,559 out of 4,252 stories contain at least one detected ambiguity, corresponding to approximately 83.7% of the corpus. This high prevalence reflects the

[†] https://gitlab.com/explore/projects/

TABLE 3 Descriptive statistics of the user story dataset.

Project	Abb.	User stories	Ambiguous stories	Resolved stories
GitLab Marketing Website	GMW	355	332	355
Gemnasium Python DB Analyzer	GPD	344	344	344
Gitleaks Analyzer	GLA	310	251	310
Hadolint Analyzer	HLA	327	311	327
ESLint Analyzer	ELA	521	489	521
Secrets Analyzer	SAL	982	778	982
Gemnasium Analyzer	GMA	424	424	424
Gemnasium DB Analyzer	GDB	285	285	285
Checkov Analyzer	CHA	420	61	420
Semgrep Analyzer	SGA	284	284	284
Total		**4252**	**3559**	**4252**

project-selection criteria, which favored repositories with sufficient ambiguity variation to support controlled before–after analysis. However, it also means that the dataset should not be interpreted as representative of all Agile repositories. In projects where user stories are already written with high clarity and consistent structure, the marginal benefit of SPERT-AR may be smaller. We therefore treat the ambiguity-heavy nature of the corpus as an important boundary condition for interpreting the reported improvements.

4.2 | Story Ambiguity Labeling

Each user story was analyzed for ambiguity across four linguistic dimensions grounded in established requirements engineering taxonomies: lexical, syntactic, semantic, and pragmatic Osama and Aref (2018). Ambiguity detection was conducted in two stages:

1. Automated detection using a GPT-4 classifier with structured prompts designed to identify vague wording, incomplete syntax, missing context, and cross-story overlap.
2. Heuristic checks using TF–IDF key-phrase extraction and SBERT-based similarity to detect internal inconsistencies and potential duplication.

Table 4 defines the four ambiguity types and provides representative examples. Table 5 summarizes their distribution across projects.

4.3 | Expert Validation Protocol

All automatically generated ambiguity labels were validated by five Agile practitioners with experience in requirements analysis. Reviewers independently confirmed or corrected the ambiguity assignments and evaluated the refined versions produced by the ambiguity resolution module. For each story, the experts assessed: (i) whether the refinement preserved the original intent, (ii) whether missing conditions or assumptions were correctly reconstructed, (iii) whether scope inflation occurred, and (iv) whether the refinement could plausibly influence practitioner interpretation of effort. Any expert-suggested story-point adjustment was recorded as an exploratory annotation only and was not used as the training or evaluation label in the main experiments. Inter-rater reliability measured using Cohen's κ was 0.87, indicating strong agreement. Disagreements were resolved through discussion, resulting in a high-quality, expert-verified resolved dataset for all subsequent experiments.

4.4 | Exploratory Analysis of Effort Interpretation

Among the 4,252 annotated stories, 3,559 contained at least one ambiguity. The resolution process generated a parallel corpus that supports direct comparison between original and clarified textual representations. In addition to validating the refined stories, experts also recorded whether clarification could plausibly change a practitioner's interpretation of effort. These expert-suggested effort changes are treated only as exploratory annotations and are not used as the target labels in the main predictive experiments. The primary evaluation continues to use the original repository-assigned story points for both original and resolved stories.

Approximately 28% of stories received an exploratory effort-adjustment flag after clarification, most often because vague or underspecified tasks became more concretely scoped. These exploratory effort-adjustment annotations are not used in any MAE, MSE, SA, statistical-significance, or ablation results reported in Section 5. Figure 5 illustrates a representative ambiguous story from the GitLab Marketing Website project VersionOne (2021). Expressions such as "correct paths" and "slightly modified build steps" created uncertainty about implementation scope; after refinement and expert review, the story was annotated as a case where practitioner interpretation of effort could change. This example is used to illustrate the practical effect of ambiguity on estimation judgment, not to redefine the primary ground-truth label used for model evaluation.

Formally, the primary resolved dataset used for model training and evaluation is defined as:

$$\mathcal{D}_{\text{resolved}} = \{(U_i^*, y_i)\}_{i=1}^{N}, \tag{22}$$

TABLE 4 Framework for identifying user story ambiguities.

Ambiguity type	Linguistic level	Requirements violation	Example violation
Vagueness	Lexical	Use of polysemous or unclear terms	"Optimize system" lacks a measurable target.
Inconsistency	Syntactic	Poor structure or grammar	Compound stories mixing unrelated tasks.
Insufficiency	Semantic	Missing context or conditions	No preconditions, assumptions, or actor roles.
Duplication	Pragmatic	Redundant or overlapping stories	Similar functionality split across stories.

TABLE 5 Ambiguity types identified across projects.

Project	Lexical	Syntactic	Semantic	Pragmatic
GMW	110	152	68	2
GPD	252	290	60	3
GLA	181	193	20	0
HLA	211	241	106	1
ELA	380	452	62	0
SAL	423	718	391	1
GMA	259	347	101	3
GDB	186	225	31	0
CHA	38	48	19	0
SGA	258	188	22	0
Total	**2298**	**2854**	**880**	**10**

FIGURE 5 Example of an ambiguous user story prior to ambiguity resolution.

where U_i^* is the expert-validated resolved story retained after the ambiguity-resolution process, and y_i is the original repository-assigned story point used as the primary ground-truth label. When experts suggested that clarification could alter effort interpretation, the corresponding exploratory annotation is stored separately as y_i^{exp} and is not used in the main accuracy evaluation. Table 6 provides examples demonstrating how ambiguity can affect interpretation and how refinements reduce unclear wording, missing context, or overlapping scope.

This dual-view dataset forms the empirical foundation for evaluating SPERT-AR, supporting controlled experiments, ablation studies, and cross-project analysis reported in Section 5. The full annotated dataset, including original, ambiguous, and refined stories, is publicly available for reproducibility.‡

‡ https://github.com/waleedunas/spert-ar-dataset

5 | EXPERIMENTAL EVALUATION

This section empirically evaluates SPERT-AR. We first describe the experimental design and baseline models, then present the metrics used to assess estimation accuracy and ambiguity-resolution quality. The subsequent subsections address four research questions on the impact of ambiguity resolution, comparison with baselines, cross-project generalization, and computational overhead, followed by an ablation study of SPERT-AR components.

Table 7 summarizes the main methodological components used in the experimental evaluation, including the dataset, feature representation, algorithms, metrics, and validation setup.

5.1 | Experimental Design

All experiments use the dual-view dataset described in Section 4, where each user story appears in both its original and resolved form. Each record includes `issuekey`, `created`, `title`, `description`, `storypoints`, and `ambiguity_types`. Resolved stories are obtained through the SPERT-AR ambiguity-resolution pipeline (Section 3), which applies GPT-4 prompts guided by detected ambiguity categories and then passes all outputs through expert validation. The original repository-assigned story points are preserved as the primary ground-truth labels in all main training and evaluation experiments. Ambiguity resolution changes only the textual representation of a story; each resolved story U_i^* remains paired with the same repository label y_i. This fixed-label design isolates the effect of textual clarification on estimation performance without introducing label changes as a confounding factor.

Expert-adjusted story points are retained only as secondary exploratory annotations. They are used to examine how clarification may influence practitioner interpretation of effort, but they are not used as the ground truth for the headline accuracy results. To reflect realistic deployment settings, the data are partitioned using a 60/20/20 split for training, validation, and testing with strict cross-project isolation; all stories from a given project appear in exactly one partition. This prevents data leakage and mirrors practical scenarios in which estimation models are applied to previously unseen projects. All experiments are repeated five times with different random seeds, and results are reported as averages.

TABLE 6 Examples of ambiguity and resolution in user stories.

Ambiguity type	Ambiguous user story snippet	Refined version
Lexical	"Improve performance of the analytics module."	"Improve the performance of the analytics module when processing user queries."
Syntactic	"As a user, I want to reset password and receive alerts."	Split into: (1) "Reset password via email." (2) "Configure email alerts for login events."
Semantic	"As a user, I want to register."	Expanded with missing steps for email verification and input validation.
Pragmatic	"Sort products by price'' and "Filter products by price'' listed separately.	"Clarify whether sorting and filtering by price should be handled as separate or combined functionality."

TABLE 7 Summary of experimental methodology.

Component	Description
Dataset description	Ten publicly available GitLab projects containing 4,252 user stories with titles, descriptions, repository-assigned story points, ambiguity labels, and expert-validated resolved versions.
Feature representation	User story titles and descriptions are concatenated and encoded using transformer-based contextual representations for SPERT and SPERT-AR; baseline models use TF–IDF vectors, averaged embeddings, or SBERT embeddings depending on the model.
Algorithms	SPERT-AR is compared with SPERT, SBERT-XG, TF-IDF-SE, and Deep-SE under identical data partitions. SPERT-AR combines ambiguity-aware refinement with reinforced transformer-based estimation.
Evaluation metrics	Prediction performance is assessed using MAE, MSE, and SA. Ambiguity-resolution quality is assessed using ARR, SRS, and EAR. Statistical reliability is examined using paired significance tests and effect sizes.
Experimental setup	Experiments use a 60/20/20 train-validation-test split with strict cross-project isolation. All models are evaluated over five random seeds, and results are reported as averages.

5.2 | Controlled Evaluation of Ambiguity Resolution

To isolate the effect of ambiguity resolution from changes in target labels or model complexity, we design the main experiment as a controlled text-only intervention. For each user story, the original version U_i and the resolved version U_i^* are paired with the same repository-assigned story point y_i. The comparison therefore evaluates whether modifying the textual clarity of the input improves prediction accuracy while holding the target label fixed. In this setting, any improvement cannot be attributed to expert-adjusted labels, because the adjusted values are excluded from the primary training and testing procedure.

This design separates three sources of possible improvement: (i) the effect of ambiguity resolution, measured by comparing models trained on original stories against the same models trained on resolved stories under identical labels; (ii) the effect of model architecture, measured by comparing SPERT-AR with non-reinforced baselines under the same resolved-text condition; and (iii) the exploratory effect of effort reinterpretation, analyzed separately using expert annotations but not included in the headline predictive results. This separation allows the study to test the central hypothesis more directly: reducing ambiguity in user-story text improves the learnability and stability of story point estimation without redefining the ground-truth effort labels.

5.3 | Baselines

We evaluate SPERT-AR against baselines at two levels: ambiguity resolution and story point estimation.

Ambiguity-resolution baselines. The GPT-4–based refinement module in SPERT-AR is compared to two alternative techniques for generating resolved stories: (i) a rule-based approach that uses manually crafted linguistic patterns to flag and modify vague or structurally incomplete expressions, and (ii) a BERT-based approach that employs a fine-tuned ambiguity classifier with token-level masking to rewrite unclear phrases. The rule-based method is brittle and highly sensitive to project-specific phrasing, whereas the BERT-based method often misses broader contextual cues and produces shallow edits. SPERT-AR integrates explicit ambiguity categorization with structured GPT-4 prompts to yield richer, context-aware rewrites. In all three cases, outputs are subject to expert validation to ensure semantic fidelity. A qualitative comparison is summarized in Table 8.

Story point estimation baselines. For effort estimation, SPERT-AR is compared against three representative models: TF-IDF-SE Tawosi et al. (2022b), which uses TF–IDF vectors with a linear regressor; Deep-SE Choetkiertikul et al. (2018), which applies averaged word embeddings and a multilayer perceptron; and SBERT-XG Yalçıner et al. (2024), which combines Sentence-BERT embeddings with an XGBoost regressor. The original SPERT model Younas et al. (2025), trained on ambiguous stories, serves as the primary learning-based baseline. SPERT-AR extends

TABLE 8 Comparison of ambiguity-resolution techniques.

Technique	Accuracy	Manual effort	Scalability	Context awareness
Rule-based	Moderate	High	Low	Low
BERT-based	High	Medium	Medium	Moderate
GPT-4 (ours)	Very high	Low (expert validation only)	High	High

SPERT by integrating ambiguity detection, resolution, and expert feedback into the estimation pipeline. All models are trained on both ambiguous and resolved datasets using identical pre-processing, data splits, and hyperparameter tuning to ensure a fair comparison.

5.4 | Evaluation Metrics

We evaluate SPERT-AR and the baselines along three dimensions: estimation accuracy, ambiguity-resolution quality, and statistical reliability.

Estimation accuracy. We report Mean Absolute Error (MAE), Mean Squared Error (MSE), and Standardized Accuracy (SA). MAE measures the average absolute deviation between predicted and true story points, while MSE penalizes larger errors more heavily. In all primary experiments, the true value is the original repository-assigned story point y_i, regardless of whether the input text is original or resolved.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{23}$$

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{24}$$

SA assesses performance relative to a random estimator Shepperd et al. (2012):

$$\text{SA} = 1 - \frac{\text{MAE}_{\text{model}}}{\text{MAE}_{\text{random}}}, \tag{25}$$

where $\text{MAE}_{\text{random}}$ denotes the expected MAE of a random predictor estimated on the same test partition. For reporting consistency, SA is expressed as a percentage by multiplying the resulting value by 100. All random-baseline values are computed separately for each test split, and all metrics are averaged over five runs with different random seeds.

Ambiguity-resolution quality. To quantify the effect of the resolution step itself, we use three metrics aligned with the formal model in Section 3: Ambiguity Reduction Rate (ARR), Semantic Retention Score (SRS), and Expert Agreement Ratio (EAR). For each story, ARR captures the proportional decrease in ambiguity score after applying the resolution operator $\mathcal{R}$; SRS measures semantic similarity between original and refined stories; EAR records the proportion of cases in which experts accept the refined story:

$$\text{ARR} = 1 - \frac{A(\hat{U}_i)}{A(U_i)}, \tag{26}$$

$$\text{SRS} = \frac{1}{N}\sum_{i=1}^{N}\text{Sim}(U_i, \hat{U}_i), \tag{27}$$

$$\text{EAR} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}\{\text{experts validate } \hat{U}_i\}. \tag{28}$$

TABLE 9 Cross-project average performance before and after ambiguity resolution using the same original repository-assigned story-point labels in both input conditions.

Model	Dataset	MAE	MSE	SA
SPERT	Ambiguous	2.10	3.07	76.59
SPERT-AR	Resolved	**1.48**	**2.20**	**94.53**
SBERT-XG	Ambiguous	2.74	4.52	70.27
	Resolved	2.34	3.84	80.37
TF-IDF-SE	Ambiguous	4.25	5.13	57.30
	Resolved	4.20	5.00	62.52
Deep-SE	Ambiguous	2.73	3.44	68.58
	Resolved	2.39	3.11	73.90

Statistical reliability. To confirm that observed differences are not due to random variation, we apply paired t-tests and Wilcoxon signed-rank tests at the 95% confidence level, and compute Cohen's *d* as a measure of effect size. Statistical results are summarized in Table 16 and discussed in Section 5.8.

5.5 | RQ1: Impact of Ambiguity Resolution

The first research question examines whether clarified user-story text is associated with improved estimation accuracy under the fixed-label protocol. Table 9 reports cross-project averages for each estimator on original and resolved inputs, using the same repository-assigned story-point labels in both conditions. All models obtain lower MAE and MSE and higher SA on resolved stories, suggesting that clearer textual inputs provide more useful estimation signals. SPERT-AR achieves the largest gain, reducing MAE from 2.10 for SPERT on original stories to 1.48 on resolved stories and increasing SA from 76.59 to 94.53. Since SPERT-AR combines several components, this result should be interpreted together with the ablation study rather than attributed solely to ambiguity resolution. In particular, the ablation in Section 5.9 shows that applying GPT-4 rewriting to the unchanged SPERT estimator (variant A2) already accounts for a substantial share of the gain under identical labels, indicating that a meaningful portion of the headline improvement stems from clarified input text rather than from the reinforced architecture alone.

Figure 6 shows the MAE change before and after resolution. SPERT-AR exhibits the largest absolute improvement, but simpler baselines also benefit from clarified stories. Because the labels are held constant, these

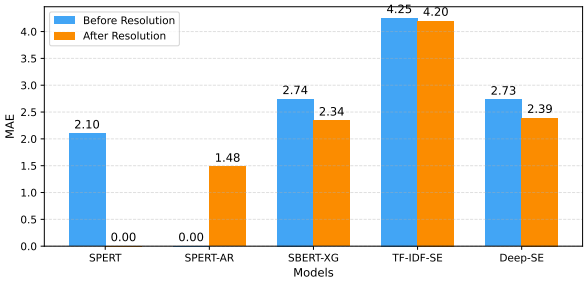

FIGURE 6 Comparison of MAE before and after ambiguity resolution across models.

TABLE 10 Ambiguity-resolution metrics per project.

Project	ARR ↑	SRS ↑	EAR ↑
GitLab Marketing Website (GMW)	0.62	0.93	0.85
Gemnasium Python DB Analyzer (GPD)	0.68	0.94	0.89
Gitleaks Analyzer (GLA)	0.63	0.91	0.87
Hadolint Analyzer (HLA)	0.60	0.92	0.83
ESLint Analyzer (ELA)	0.67	0.93	0.88
Secrets Analyzer (SAL)	0.70	0.90	0.84
Gemnasium Analyzer (GMA)	0.65	0.91	0.85
Gemnasium DB Analyzer (GDB)	0.66	0.92	0.86
Checkov Analyzer (CHA)	0.59	0.94	0.88
Semgrep Analyzer (SGA)	0.64	0.91	0.84
Cross-project average	**0.64**	**0.92**	**0.86**

gains are not explained by revised effort values. The magnitude of improvement, however, should be interpreted in light of the ambiguity-rich dataset composition.

The ambiguity-resolution metrics further substantiate these findings. Table 10 reports ARR, SRS, and EAR per project. On average, SPERT-AR reduces ambiguity by 64% while maintaining a high SRS of 0.92 and EAR of 0.86, indicating that most refined stories are both measurably clearer and semantically faithful to the originals.

To illustrate how these ambiguity-resolution metrics relate to practical estimation, consider a story with vague performance wording such as "Improve performance of the analytics module." In its original form, the story does not specify the target operation, performance threshold, or measurement context, making it difficult for estimators and learning models to infer effort-related cues. After resolution, the story is rewritten as "Reduce query latency in the analytics dashboard from 3 s to under 1 s," which makes the expected change more concrete while preserving the original performance-improvement intent. This type of refinement explains why ARR and SRS must be interpreted jointly: the goal is not only to reduce ambiguity, but to do so without changing the underlying functional scope.

From a practical perspective, such refinements can support backlog discussions by making hidden assumptions visible before story point estimation occurs. Product owners and developers can then confirm whether the clarified target is acceptable, revise it if necessary, or defer estimation until missing information is available. Thus, the improvement

TABLE 11 Impact of ambiguity resolution techniques on story point estimation (average across 10 projects).

Method	MAE (↓)	MSE (↓)	SA (↑)
No resolution (SPERT baseline)	2.10	3.07	76.59%
Rule-based resolution	1.86	2.75	81.70%
BERT-based classifier	1.67	2.49	86.00%
LLM-based refinement (SPERT-AR)	**1.48**	**2.20**	**94.53%**

TABLE 12 Performance comparison between SPERT-AR and ambiguity classifiers.

Model	MAE (↓)	MSE (↓)	SA (↑)
SPERT-AR (proposed)	**1.48**	**2.20**	**94.53%**
BERT-classifier	1.86	3.45	72.43%
TF-IDF classifier	2.49	4.86	61.30%

in estimation accuracy reported in Table 9 should be understood not only as a model-level gain, but also as evidence that clearer story formulations provide more stable inputs for both automated estimators and human planning discussions.

Table 11 compares alternative resolution strategies averaged over all projects. SPERT-AR LLM-based refinement outperforms rule-based and BERT-based techniques on all three metrics. Table 12 contrasts SPERT-AR with models that only detect ambiguity without resolving it. Detection alone yields modest gains, but the full pipeline that explicitly reduces ambiguity and feeds refined stories into the estimator delivers the largest improvements, indicating that resolution must be integrated into the estimation workflow rather than treated as a detached pre-processing step.

5.6 | RQ2: Comparison with Baselines

The second research question investigates how SPERT-AR compares with traditional and modern estimators at the project level. Table 13 reports MAE, MSE, and SA for all models when trained on resolved stories. On ambiguous inputs (table omitted for brevity), SPERT already outperforms TF-IDF-SE, Deep-SE, and SBERT-XG, confirming the benefit of reinforced transformer-based estimation. After ambiguity resolution, SPERT-AR consistently achieves the lowest MAE and highest SA across all ten projects.

Projects with higher initial ambiguity, such as Gemnasium Analyzer (GMA) and Secrets Analyzer (SAL), show particularly large relative improvements, reflecting stronger noise reduction by the resolution pipeline. The aggregated comparison in Table 14 shows that, on average, SPERT-AR reduces MAE by 29.52% and improves SA by 23.41% relative to SPERT trained on ambiguous stories.

TABLE 13 Model evaluation metrics on resolved stories across all projects.

Project	Abb.	Model	MAE	MSE	SA (%)	Project	Abb.	Model	MAE	MSE	SA (%)
GitLab Marketing Website	GMW	SPERT-AR	**1.13**	**1.20**	**97.79**	Secrets Analyzer	SAL	SPERT-AR	**1.58**	**4.43**	**84.70**
		SBERT-XG	1.90	3.70	75.24			SBERT-XG	1.99	6.27	69.75
		TF-IDF-SE	4.97	4.56	46.29			TF-IDF-SE	4.89	6.12	62.05
		Deep-SE	1.80	3.48	65.94			Deep-SE	2.09	4.09	71.76
Gemnasium Python DB Analyzer	GPD	SPERT-AR	**0.69**	**1.38**	**93.72**	Gemnasium Analyzer	GMA	SPERT-AR	**3.58**	**4.37**	**88.36**
		SBERT-XG	1.53	1.60	80.07			SBERT-XG	4.53	6.77	79.02
		TF-IDF-SE	3.40	5.54	55.65			TF-IDF-SE	7.18	9.04	66.91
		Deep-SE	1.62	1.79	80.73			Deep-SE	6.69	7.13	71.76
Gitleaks Analyzer	GLA	SPERT-AR	**1.17**	**1.26**	**94.43**	Gemnasium DB Analyzer	GDB	SPERT-AR	**1.93**	**2.69**	**95.09**
		SBERT-XG	2.34	2.50	81.72			SBERT-XG	2.32	4.04	90.45
		TF-IDF-SE	2.81	2.94	55.65			TF-IDF-SE	4.87	5.76	64.96
		Deep-SE	1.82	1.81	76.32			Deep-SE	2.64	3.67	71.19
Hadolint Analyzer	HLA	SPERT-AR	**1.88**	**2.32**	**91.35**	Checkov Analyzer	CHA	SPERT-AR	**1.47**	**2.93**	**92.81**
		SBERT-XG	2.73	3.00	78.15			SBERT-XG	2.70	3.67	90.85
		TF-IDF-SE	3.51	4.20	64.14			TF-IDF-SE	3.71	4.62	77.74
		Deep-SE	2.33	3.38	66.24			Deep-SE	2.76	3.64	80.27
ESLint Analyzer	ELA	SPERT-AR	**1.05**	**2.76**	**95.52**	Semgrep Analyzer	SGA	SPERT-AR	**1.31**	**1.61**	**93.36**
		SBERT-XG	1.20	3.46	78.05			SBERT-XG	2.02	2.11	86.68
		TF-IDF-SE	2.39	5.26	64.96			TF-IDF-SE	2.27	3.22	69.73
		Deep-SE	1.48	4.57	66.79			Deep-SE	2.67	2.55	74.32

TABLE 14 Comparison of SPERT and SPERT-AR on original and resolved user stories (average across 10 projects).

Model	MAE (↓)	MSE (↓)	SA (↑)
SPERT (ambiguous)	2.10	3.07	76.59%
SPERT-AR (resolved)	**1.48**	**2.20**	**94.53%**
Improvement	*29.52%*	*28.34%*	*23.41%*

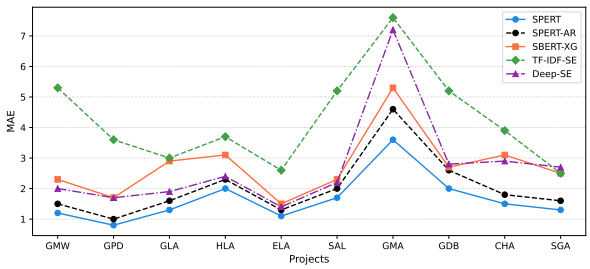

FIGURE 7 Cross-project performance of different models on original and resolved user-story inputs.

5.7 | RQ3: Cross-Project Generalization

The third research question focuses on robustness under domain shift. We adopt a cross-project validation setup in which models are trained on a subset of projects and evaluated on previously unseen ones. This configuration mirrors realistic Agile practice, where estimation tools are often applied across teams and domains with different vocabularies and conventions.

Figure 7 summarizes cross-project performance for ambiguous and resolved datasets. SPERT-AR consistently attains the lowest error and exhibits the most stable behavior on held-out projects. The gains stem from two factors: reinforcement learning, which adapts the estimation policy to heterogeneous effort distributions, and disambiguated user stories, which provide clearer and more transferable effort signals. Across all models, training on resolved stories improves cross-project performance compared to training on ambiguous ones, confirming that LLM-assisted ambiguity resolution benefits not only in-domain accuracy but also transferability. Traditional baselines such as TF-IDF-SE and SBERT-XG show weaker transfer, reflecting limited ability to cope with linguistic variability without explicit resolution.

5.8 | RQ4: Computational Overhead and Statistical Robustness

The final research question examines the computational and operational trade-offs introduced by ambiguity resolution, together with the statistical robustness of the observed improvements. SPERT-AR adds an upfront refinement stage because ambiguity detection and resolution require LLM inference and expert validation before model training. Its practical value therefore depends not only on predictive accuracy, but

TABLE 15 Computational resource comparison.

Model	Ambiguous dataset (hours)	Resolved dataset (hours)
SPERT-AR	12.0	10.0
SBERT-XG	6.0	5.0
TF-IDF-SE	4.0	3.5
Deep-SE	8.0	7.0

also on whether this additional effort can be managed within realistic backlog-refinement workflows.

Table 15 reports average end-to-end training time for each model on original and resolved datasets. SPERT-AR has the highest computational cost because it includes GPT-4-based refinement and reinforcement optimization. Much of this cost, however, is incurred offline. Once a story has been clarified and validated, the resolved version can be cached and reused unless the story text changes. The ambiguity score can also act as a gate, so that LLM-based rewriting is applied only to stories above a selected ambiguity threshold.

Operationally, SPERT-AR is better suited to scheduled backlog refinement than to real-time estimation during sprint planning. Teams can run the ambiguity-resolution stage before refinement or release-planning meetings and then use the clarified stories during estimation. A hybrid deployment is also possible: low-risk refinements can be accepted after semantic-similarity checks, while high-risk cases are routed to human reviewers.

The values in Table 15 should therefore be interpreted as full experimental pipeline costs, not as the cost of generating a single prediction after deployment. In practical use, the most expensive stage is the initial refinement of ambiguous stories. Once the clarified corpus is available, subsequent estimation can be performed using the trained estimator without repeatedly invoking GPT-4 for unchanged stories. This distinction is important because SPERT-AR is designed as an ambiguity-aware refinement and estimation workflow, not as a real-time LLM-only predictor.

To verify that performance differences are statistically reliable, we apply Wilcoxon signed-rank tests to paired samples before and after ambiguity resolution and compute Cohen's d for effect size. Table 16 shows that SPERT-AR achieves highly significant improvements ($p < 0.01$) with large effect sizes ($d > 1.0$) across all metrics. Most baselines also show statistically significant gains, although their effect sizes are smaller. These results indicate that the benefits of ambiguity-aware training are not artifacts of random fluctuation.

5.9 | Ablation Study

To separate the contribution of ambiguity-related components from model complexity, we compare four controlled variants. A1 is the original SPERT baseline trained on original user-story text. A2 keeps the SPERT estimator unchanged but applies GPT-4 rewriting without the formal ambiguity model. A3 keeps the estimator unchanged but adds the formal ambiguity model without GPT-4 rewriting. A4 is the complete SPERT-AR pipeline, combining formal ambiguity modeling, GPT-4-based refinement, expert validation, and reinforced estimation. Table 17 reports normalized MAE, MSE, and SA across the ten projects, with A1 fixed at 1.000 for MAE and MSE and 0.000 for SA.

TABLE 16 Statistical analysis of performance improvements (before vs. after ambiguity resolution).

Metric	Model	p-value	Effect size (Cohen's d)
MAE	SPERT-AR	0.0012	1.23
	SBERT-XG	0.0031	1.07
	TF-IDF-SE	0.0480	0.56
	Deep-SE	0.0520	0.49
MSE	SPERT-AR	0.0023	1.15
	SBERT-XG	0.0064	0.98
	TF-IDF-SE	0.0350	0.62
	Deep-SE	0.0590	0.45
SA	SPERT-AR	0.0008	1.45
	SBERT-XG	0.0042	1.20
	TF-IDF-SE	0.0410	0.60
	Deep-SE	0.0650	0.40

TABLE 17 Ablation study results across ten projects (normalized values).

Model variant	MAE (↓)	MSE (↓)	SA (↑)
A1: SPERT (baseline)	1.000	1.000	0.000
A2: SPERT + GPT-4	0.812	0.752	0.174
A3: SPERT + formal model	0.774	0.723	0.202
A4: SPERT-AR (full)	**0.705**	**0.681**	**0.234**

The purpose of this ablation is not to claim that ambiguity resolution is the sole source of improvement, but to examine whether ambiguity-aware components provide measurable gains beyond the SPERT baseline. Because SPERT-AR includes interacting mechanisms, the results should be interpreted as evidence that ambiguity-related refinement contributes to the observed gains while acknowledging possible interaction effects among clearer inputs, expert validation, and reinforcement-based learning.

The normalized results show that both GPT-4-based rewriting and formal ambiguity modeling improve performance relative to the SPERT baseline, while the full SPERT-AR variant achieves the strongest results. Figure 8 visualizes these relative gains. The formal ambiguity model structures ambiguity signals, GPT-4 rewriting improves textual clarity, and expert validation helps preserve scope and intent. However, the full improvement should be interpreted as the combined effect of ambiguity-aware refinement and reinforced estimation rather than as the isolated effect of a single component.

Overall, the ablation study supports the contribution of ambiguity-aware components to improved estimation performance, while recognizing that interactions among input refinement, expert validation, and model learning cannot be fully eliminated.

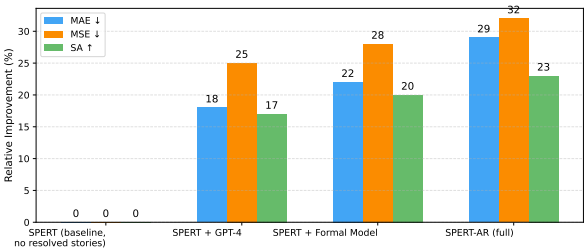

FIGURE 8 Relative performance gains from SPERT-AR components in the ablation study.

6 | DISCUSSION

6.1 | Interpretation of Results

The results indicate that linguistic ambiguity affects the learnability of story-point estimation models. Across ten projects, SPERT-AR trained on clarified stories reduces MAE by 29.52% and improves SA by 23.41% compared with SPERT trained on original inputs (Table 14). These improvements are obtained under a fixed-label evaluation protocol in which both original and resolved stories are paired with the same repository-assigned story points. Therefore, the results should be interpreted as evidence that ambiguity-aware textual refinement improves the model's ability to recover historical effort estimates, not as evidence that ambiguity resolution changes the true development effort of a story.

This interpretation requires two qualifications. First, the improvement reflects the combined effect of ambiguity-aware refinement and the SPERT-AR estimation architecture; the ablation study therefore helps separate the contribution of individual components. Second, because the dataset is intentionally ambiguity-rich, the observed gains may be stronger than in projects where user stories are already clear and consistently structured. The exploratory expert annotations further show that clarification can make hidden assumptions and scope boundaries more explicit, but these annotations are not used as primary ground truth. Repository-assigned story points remain the main labels, while expert-suggested effort reinterpretations are treated only as secondary evidence for future work on human estimation behavior.

Ambiguity-resolution metrics support this interpretation. SPERT-AR achieves an ARR of 0.64, an SRS of 0.92, and an EAR of 0.86 on average (Table 10), indicating that refinements generally increase clarity while preserving original intent. The ablation study (Section 5.9) confirms that the ambiguity model, GPT-4 rewriting, and expert validation each contribute to the final performance, with the complete pipeline delivering the strongest results. Cross-project evaluation (Figure 7) further shows that SPERT-AR remains more accurate and less variable than competing models on unseen repositories. Overall, the evidence supports the claim that ambiguity-aware estimation is feasible and empirically effective, while keeping the scope of the claim limited to prediction under unchanged repository labels.

6.2 | Practical Implications

The findings suggest that ambiguity handling can improve planning support, but SPERT-AR should be understood as a decision-support workflow rather than an autonomous estimation system. Its most appropriate use is offline backlog refinement before sprint planning. The framework can analyze new or updated stories, identify ambiguity types, generate clarification suggestions, and present them to product owners, business analysts, or senior developers for review. The final estimate remains a team decision; SPERT-AR's role is to reduce textual uncertainty before estimation discussions begin.

In practice, SPERT-AR could be integrated into platforms such as GitLab, Jira, or Azure DevOps through a plugin or external service. Low-ambiguity stories may proceed directly to estimation, while high-ambiguity stories can be queued for LLM-based refinement and human validation. The system can store original and clarified versions as linked artifacts, together with ambiguity types and reviewer decisions, preserving traceability across the refinement process.

The main operational costs arise from repeated LLM calls and expert validation. These costs can be reduced by applying rewriting only to stories above an ambiguity threshold, caching clarified stories, batching low-priority items before planning meetings, routing only low-confidence or high-impact cases to reviewers, and using private or domain-adapted models when data privacy or recurring cost is a concern. Under such a workflow, expert effort is concentrated on stories where ambiguity is most likely to affect planning.

These deployment considerations also define the limits of the framework. Teams with inconsistent story-point practices, limited reviewer availability, strict data-governance constraints, or very large backlogs may require additional customization. Thus, SPERT-AR does not eliminate human estimation effort; it provides a structured mechanism for making ambiguity reduction more traceable, repeatable, and measurable before estimation.

6.3 | Limitations and Future Directions

Several limitations define the scope of this work. The dataset consists of ten open-source GitLab projects, mainly from security and analysis domains. Other settings, including embedded systems, finance, healthcare, or enterprise software, may follow different story-writing practices, ambiguity patterns, and estimation cultures. Replication with proprietary datasets, alternative estimation scales such as T-shirt sizes or function points, and varied tool chains is therefore needed to assess generalizability.

The ambiguity-resolution process also introduces practical overhead. GPT-4-based refinement improves textual clarity, but repeated LLM calls may increase cost, latency, and dependence on external services. Expert validation improves semantic fidelity but can become difficult to scale for very large backlogs. Future work should quantify cost-control

strategies such as threshold-based rewriting, caching, batching, selective expert review, and the use of private, domain-adapted, or smaller local models in industrial settings.

Data governance and privacy constraints may also affect deployment. Industrial user stories can contain sensitive business logic, client information, security requirements, or commercially confidential details. Organizations may therefore be unable to send backlog items to external LLM APIs. Future versions of SPERT-AR should investigate privacy-preserving deployments, including on-premise language models, anonymization pipelines, secure logging, and integration with enterprise access-control policies.

Finally, SPERT-AR operates on static snapshots of user stories and does not incorporate temporal aspects such as story evolution, sprint dynamics, team velocity, developer availability, or historical estimation adjustments. It also does not use auxiliary artifacts such as comments, discussion threads, linked merge requests, or code changes, which may contain additional effort cues. Future studies should evaluate SPERT-AR in live Agile teams to measure not only prediction accuracy but also review effort, refinement acceptance rate, perceived clarity, planning confidence, and impact on estimation discussions.

7 | THREATS TO VALIDITY

This section outlines potential threats that may influence the interpretation of the results and describes the steps taken to mitigate their impact.

Internal Validity. Internal validity concerns whether the observed improvements can be attributed to SPERT-AR rather than to uncontrolled experimental factors. To reduce confounding effects, all evaluated models used identical preprocessing pipelines, data partitions, and hyperparameter tuning procedures. Strict cross-project isolation was also enforced so that stories from the same project did not appear in multiple partitions. A remaining source of subjectivity arises from expert validation of ambiguity refinements. To reduce this risk, multiple Agile practitioners independently reviewed the refinements, inter-rater agreement was measured using Cohen's κ (0.87), and disagreements were resolved through discussion. Nevertheless, some human judgment remains inherent in the validation process.

Construct Validity. Construct validity concerns whether the study accurately measures effort estimation quality and ambiguity reduction. MAE, MSE, and SA are standard metrics in software effort estimation and provide complementary views of prediction error, large-error sensitivity, and performance relative to a random baseline. In the primary experiments, original repository-assigned story points are used as ground-truth labels for both original and resolved stories. This design avoids conflating textual clarification with post-hoc label revision and isolates the effect of clarified input on predictive performance. However, repository-assigned story points are historical team judgments rather than objective measures of actual development effort. They may reflect local estimation habits, team velocity, negotiation dynamics, or project-specific conventions. Therefore, the results should be interpreted as improvements in predicting historical story-point assignments, not as direct proof of improved prediction of actual development time or cost.

Ambiguity is modeled across four linguistically grounded categories: lexical, syntactic, semantic, and pragmatic. Although this taxonomy is consistent with requirements-engineering literature, it may not capture all domain-specific or organization-specific ambiguity forms. Similarly, ARR, SRS, and EAR measure ambiguity reduction and semantic preservation, but they do not fully capture human factors such as developer confidence, negotiation quality, or shared understanding during sprint planning. Expert-suggested effort reinterpretations are therefore treated as exploratory annotations rather than primary ground-truth labels.

External Validity. External validity concerns generalizability beyond the studied setting. The evaluation dataset consists of ten open-source GitLab projects, with a concentration in security and static-analysis tools. Industrial environments may differ in development practices, estimation scales, backlog platforms, team culture, and domain complexity. These differences could influence both ambiguity prevalence and model behavior. Consequently, the reported improvements should be interpreted within the context of the studied dataset and require further validation using proprietary industrial datasets and projects from different application domains.

A further external-validity concern is the high prevalence of ambiguous stories in the corpus. Since 3,559 out of 4,252 stories contain at least one detected ambiguity, the dataset provides a useful stress test for SPERT-AR but may also amplify the apparent benefit of ambiguity-aware refinement. Projects with clearer story-writing conventions or stricter backlog-quality practices may exhibit smaller performance gains.

Statistical Conclusion Validity. Statistical conclusion validity concerns whether the reported improvements are statistically reliable. To improve robustness, experiments were repeated using multiple random seeds, and both paired t-tests and Wilcoxon signed-rank tests were applied at the 95% confidence level. Effect sizes using Cohen's *d* accompany the significance tests (Table 16), and the results indicate large and statistically significant improvements for SPERT-AR. However, the evaluation includes only ten projects, and some repositories may share contributors or development practices, which can introduce partial dependencies and optimistic variance estimates. The results should therefore be interpreted with appropriate caution.

8 | CONCLUSION

This paper introduced SPERT-AR, a framework that combines formal ambiguity modeling, LLM-based refinement, expert validation, and reinforcement learning for story point estimation in Agile software development. Rather than assuming that user stories are already clear and

complete, SPERT-AR treats ambiguity as an input-quality issue that can weaken the textual signals used by learning-based estimators.

Experiments on ten GitLab projects show that clarified stories improve estimation performance under a fixed-label evaluation protocol. SPERT-AR reduces MAE by 29.52% and improves SA by 23.41% relative to SPERT trained on original inputs, while preserving repository-assigned story points as the primary evaluation labels. Ambiguity-resolution metrics further indicate that the refinement process reduces linguistic uncertainty while maintaining semantic fidelity. These results suggest that ambiguity-aware refinement can support more stable estimation under the ambiguity-rich conditions studied in this paper.

The findings should be interpreted as evidence that clearer user-story representations can improve learning-based story point estimation, not as a claim that ambiguity resolution alone determines estimation accuracy or changes the true development effort of a story. Future work should evaluate SPERT-AR in broader industrial settings, with different estimation scales, backlog-management tools, privacy constraints, and team practices, to assess how ambiguity-aware refinement affects both model performance and human estimation workflows.

REFERENCES

Abo-eleneen, A., Palliyali, A. & Catal, C. (2023) The role of reinforcement learning in software testing. *Information and Software Technology*, 164, 107325. doi:10.1016/j.infsof.2023.107325.

Ali, M., Mazhar, T., Arif, Y., Al-Otaibi, S., Ghadi, Y.Y., Shahzad, T. et al. (2024) Software defect prediction using an intelligent ensemble-based model. *IEEE Access*, 12, 20376–20395. doi:10.1109/ACCESS.2024.3358201.

Ali, M., Mazhar, T., Shahzad, T., Ghadi, Y.Y., Mohsin, S.M., Akber, S.M.A. et al. (2023) Analysis of feature selection methods in software defect prediction models. *IEEE Access*, 11, 145954–145974. doi:10.1109/ACCESS.2023.3343249.

Alomari, K. & Elazhary, H. (2018) Ambiguity prevention in software requirement specification using finite state machines. *International Journal of Advanced Computer Science and Applications*, 9(7), 320–326. doi:10.14569/IJACSA.2018.090745.

Amna, A.R. & Poels, G. (2022) Ambiguity in user stories: A systematic literature review. *Information and Software Technology*, 145, 106824. doi:https://doi.org/10.1016/j.infsof.2022.106824.
URL https://www.sciencedirect.com/science/article/pii/S0950584922000040

Antoniou, C. & Bassiliades, N. (2024) A tool for requirements engineering using ontologies and boilerplates. *Automated Software Engineering*, 31(1), 5. doi:10.1007/s10515-023-00403-y.

Ashfaq, A. & Bajwa, I.S. (2021) Bridging natural language requirements and formal logic using controlled natural language (sbvr). *Automated Software Engineering*, 28, 1–25. doi:10.1007/s10515-020-00289-9.

Butt, S.A., Ercan, T., Binsawad, M., Ariza-Colpas, P.P., Diaz-Martinez, J., Pineres-Espitia, G.D. et al. (2023) Prediction based cost estimation technique in agile development. *Advances in Engineering Software*, 16. doi:10.1016/j.advengsoft.2022.103329.

Choetkiertikul, M., Dam, H.K., Tran, T. & Ghose, A. (2018) A deep learning model for estimating story points. *IEEE Transactions on Software Engineering*, 45(7), 637–656. doi:10.1109/TSE.2018.2808394.

da Silva Neo, G., Moura, J.A.B., Almeida, H.O., da Silva Neo, A.V.B. & Júnior, O.d.G.F. User story tutor (ust) to support agile software developers. In: *CSEDU (2), 2024*, pp. 51–62.

Dar, S., Bajwa, I.S., Khan, M.U. & Hussain, T. (2024) Gamify4lexamb: A gamification-based approach for reducing lexical ambiguity in software requirements. *IEEE Access*, 12, 17789–17805. doi:10.1109/ACCESS.2024.3356781.

Devlin, J., Chang, M.W., Lee, K. & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2019*. : Association for Computational Linguistics, pp. 4171–4186.

Edison, H., Wang, X. & Conboy, K. (2021) Comparing methods for large-scale agile software development: A systematic literature review. *IEEE Transactions on Software Engineering*, 48(8), 2709–2731. doi:10.1109/TSE.2021.3069039.

Fabbrini, F., Fusani, M., Gnesi, S. & Lami, G. An automatic quality evaluation for natural language requirements. In: *Proceedings of the IEEE International Conference on Software—NASA Goddard, 2001*. : IEEE, pp. 1–10.

Faris, H.M., Faris, H., Ibrahim, S., Aloqaily, M. & Otair, M. (2021) Text analytics and story points estimation using nlp techniques. *IEEE Access*, 9, 73170–73185. doi:10.1109/ACCESS.2021.3079104.

Ferrucci, F., Sarro, F. & Mendes, E. (2019) An empirical comparison of model-based and data-driven effort estimation methods. *Empirical Software Engineering*, 24(3), 1565–1603. doi:10.1007/s10664-018-9653-2.

Fichtinger, A. & Lübke, D. On the pragmatic ambiguity in natural language requirements. In: *International Working Conference on Requirements Engineering: Foundation for Software Quality, 2015*. : Springer, pp. 183–190.

Foschini, L. (2021) Project management in the consultancy sector: Comparing waterfall and agile approaches. Ph.D. thesis, Politecnico di Torino, Torino, Italy.

Fu, M. & Tantithamthavorn, C. (2022) Gpt2sp: A transformer-based agile story point estimation approach. *IEEE Transactions on Software Engineering*, 49, 611–625. doi:10.1109/TSE.2022.3158252.

Fujimoto, S., van Hoof, H. & Meger, D. Addressing function approximation error in actor-critic methods. In: *International Conference on Machine Learning*. *PMLR, 2018*, pp. 1587–1596.

Ghadi, Y.Y., Mazhar, T., Al Shloul, T., Shahzad, T., Salaria, U.A., Ahmed, A. et al. (2024) Machine learning solutions for the security of wireless sensor networks: A review. *IEEE Access*, 12, 12699–12719. doi:10.1109/ACCESS.2024.3355312.

Group, T.S. (2018) Chaos report 2018. *The Standish Group International,*.
URL https://www.standishgroup.com/sample_research_files/chaos_report_2018.pdf

Gurung, G., Shah, R. & Jaiswal, D. (2020) Software development life cycle models - a comparative study. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 6, 30–37. doi:10.32628/CSEIT206410.

Iqbal, M., Ijaz, M., Mazhar, T., Shahzad, T., Abbas, Q., Ghadi, Y.Y. et al. (2024) Exploring issues of story-based effort estimation in agile software development (asd). *Science of Computer Programming*, 236, 103114. doi:10.1016/j.scico.2024.103114.

Kapur, R. & Sodhi, B. (2022) Oss effort estimation using software features similarity and developer activity-based metrics. *ACM Transactions on Software Engineering and Methodology*, 31(2), 1–45. doi:10.1145/3485819.

Khan, S., Mazhar, T., Shahzad, T., Waheed, W., Waheed, A., Saeed, M.M. et al. (2024) Optimizing deep neural network architectures for renewable energy forecasting. *Discover Sustainability*, 5(394). doi:10.1007/s43621-024-00615-6.

Mahmood, Y., Kama, N., Azmi, A., Khan, A.S. & Ali, M. (2021) Software effort estimation accuracy prediction of machine learning techniques: A systematic performance evaluation. *Software: Practice and Experience*, 52(1), 39–65. doi:10.1002/spe.3009.

Malgonde, O. & Chari, K. (2019) An ensemble-based model for predicting agile software development effort. *Empirical Software Engineering*, 24, 1017–1055. doi:10.1007/s10664-018-9647-0.

Mazhar, T., Talpur, D.B., Al Shloul, T., Ghadi, Y.Y., Haq, I., Ullah, I. et al. (2023) Analysis of iot security challenges and its solutions using artificial intelligence. *Brain Sciences*, 13(4), 683. doi:10.3390/brainsci13040683.

Moharil, A. & Sharma, A. Identification of intra-domain ambiguity using transformer-based machine learning. In: *Proceedings of the 1st International Workshop on Natural Language-Based Software Engineering. NLBSE '22, 2023*. New York, NY, USA: Association for Computing Machinery, p. 51–58.
URL https://doi.org/10.1145/3528588.3528651

OpenAI (2023) Gpt-4 technical report. *OpenAI Research*,.
URL https://openai.com/research/gpt-4

Osama, A. & Aref, M. (2018) Dara: Detecting and resolving ambiguity in requirements specifications. *International Journal of Computer Applications*, 181(26), 42–50. doi:10.5120/ijca2018917815.

Pargaonkar, S. (2023) A comprehensive research analysis of software development life cycle (sdlc) agile & waterfall model advantages, disadvantages, and application suitability in software quality engineering. *International Journal of Scientific Research Publications (IJSRP)*, 13, 345–358. doi:10.29322/IJSRP.13.08.2023.p14015.

Poženel, M., Fürst, L., Vavpotič, D. & Hovelja, T. (2023) Agile effort estimation: Comparing the accuracy and efficiency of planning poker, bucket system, and affinity estimation methods. *International Journal of Software Engineering and Knowledge Engineering*, 33(11n12), 1923–1950. doi:10.1142/S021819402350064X.

Putri, A.Y.P. & Subriadi, A.P. (2019) Software cost estimation using function point analysis. *IPTEK Journal of Proceedings Series*,(1), 79–83. doi:10.12962/j23546026.y2019i1.5115.

Rahmawati, D., Hermawan, A., Pratama, I. & Sari, R.F. (2025) Ambitrus: A formal framework for detecting ambiguities in agile user stories. *Journal of Systems and Software*, 209, 112011. doi:10.1016/j.jss.2024.112011.

Rashid, C.H., Shafi, I., Khattak, B.H.A., Safran, M., Alfarhood, S. & Ashraf, I. (2025) Ann-based software cost estimation with input from cocomo: Cann model. *Alexandria Engineering Journal*, 113, 681–694. doi:10.1016/j.aej.2024.11.042.

Rathore, S. & Kumar, D. (2019) Story point estimation using machine learning: A case study. *International Journal of Computer Applications*, 178, 10–14. doi:10.5120/ijca2019918612.

Reimers, N. & Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, 2019*. : Association for Computational Linguistics, pp. 3982–3992.

Sánchez-García, Á.J., González-Hernández, M.S., Cortés-Verdín, K. & Pérez-Arriaga, J.C. (2024) Software estimation in the design stage with statistical models and machine learning: An empirical study. *Mathematics*, 12(7), 1058. doi:10.3390/math12071058.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017) *Proximal policy optimization algorithms*. arXiv preprint arXiv:1707.06347.

Shepperd, M., MacDonell, S.G., Kadoda, G. & Kitchenham, B. (2012) Evaluating prediction systems in software project estimation. *Information and Software Technology*, 54(8), 820–827.

Tawosi, V., Al-Subaihin, A., Moussa, R. & Sarro, F. A versatile dataset of agile open source software projects. In: *Proceedings of the 19th International Conference on Mining Software Repositories, 2022a*, Pittsburgh, PA, USA, pp. 707–711.

Tawosi, V., Moussa, R. & Sarro, F. (2022) Agile effort estimation: Have we solved the problem yet? insights from a replication study. *IEEE Transactions on Software Engineering*, 49, 2677–2697. doi:10.1109/TSE.2022.3228739.

Tran, T.N., Tran, H.T. & Nguyen, Q.N. Leveraging ai for enhanced software effort estimation: A comprehensive study and framework proposal. In: *2023 International Conference on the Cognitive Computing and Complex Data (ICCD), 2023*. : IEEE, pp. 284–289.

Venkataraman, R. & Pinto, J. (2023) *Cost and Value Management in Projects*. Hoboken, NJ, USA: John Wiley & Sons.

VersionOne (2021) 14th annual state of agile report. *VersionOne*,.Available at: https://stateofagile.com.

Wu, H.C., Luk, R.W.P., Wong, K.F. & Kwok, K.L. (2008) Interpreting tf-idf term weights as making relevance decisions. *ACM Transactions on Information Systems (TOIS)*, 26(3), 1–37. doi:10.1145/1361684.1361686.

Yalçıner, B., Dinçer, K., Karaçor, A.G. & Efe, M.Ö. (2024) Enhancing agile story point estimation: Integrating deep learning, machine learning, and natural language processing with sbert and gradient boosted trees. *Applied Sciences*, 14(16), 7305. doi:10.3390/app14167305.

Younas, W., Chen, R., Zhao, J., Iqbal, T., Sharaf, M. & Imran, A. (2025) Spert: Reinforcement learning-enhanced transformer model for agile story point estimation. *International Journal of Software Engineering and Knowledge Engineering*, 35(03), 293–325. doi:10.1142/S0218194025500044.

Zhang, J., Huang, F. & Ding, C. (2019) Learning to estimate story points: A comparison of classical and deep learning methods. *Information and Software Technology*, 107, 78–89. doi:10.1016/j.infsof.2018.10.008.