An Unified and Explainable Framework for Detecting and Resolving Linguistic Ambiguity Using Transformer-Based Models

Mr. Akshaykumar K. Bhore
Student, Department of Computer Science and Engineering (Data Science)
D.Y. Patil Agriculture and Technical University, Talsande
Mr. Somanath J. Salunkhe
Department of Computer Science and Engineering
D.Y. Patil Agriculture and Technical University, Talsande

ABSTRACT: Linguistic ambiguity presents a persistent challenge in natural language processing. Existing systems typically address a single ambiguity type, operate on monolingual input, and provide no interpretable justification for their output. This paper presents a unified, modular system for detecting and resolving five categories of linguistic ambiguity -- lexical, syntactic, semantic, coreference, and pragmatic -- using transformer-based language models. The architecture combines a Python/FastAPI backend with XLM-RoBERTa and BERT contextual encoders, a fairness-aware probability adjustment module, and an explainability engine that generates per-token importance scores. A React frontend renders interactive attention heatmaps, token importance bar charts, and force-directed relationship graphs. Multilingual support, including Devanagari Hindi and Hinglish code-mixed text, is built into the preprocessing layer. Qualitative evaluation across all five ambiguity types confirms correct classification and disambiguation for both English and code-mixed inputs, addressing several gaps identified in the prior literature.

Keywords: Linguistic Ambiguity, Transformer Models, NLP, Explainability, Multilingual, Word Sense Disambiguation, XLM-RoBERTa, FastAPI

1. INTRODUCTION

Natural language is inherently ambiguous. A single utterance can carry multiple valid interpretations at lexical, syntactic, semantic, coreference, or pragmatic levels, and human comprehension resolves these using contextual, cultural, and situational cues. Automated systems have historically struggled with the same task, particularly when inputs span multiple languages or contain code-mixed text.

Classical approaches to ambiguity resolution relied on handcrafted rule-based systems and lexical resources such as WordNet (Fellbaum, 1998). Statistical classifiers trained on annotated corpora improved generalization but depended on manual feature engineering and could not model long-range contextual dependencies (Raganato et al., 2017). The introduction of the Transformer architecture (Vaswani et al., 2017) and pre-trained models such as BERT (Devlin et al., 2019) produced a step change in context-sensitive language understanding. Bevilacqua and Navigli (2020) showed that fine-tuned BERT surpasses 80% F1 on standard word sense disambiguation benchmarks, a threshold previously considered out of reach for automated methods.

Despite these advances, three principal gaps remain in the literature. Most systems handle only one ambiguity type, typically lexical word sense disambiguation. Multilingual and code-mixed settings are poorly served, particularly for South Asian languages (Kakwani et al., 2020; Scarlini et al., 2020). Interpretable outputs are absent from nearly all existing systems (Zhao et al., 2024), and no reviewed system integrates ambiguity detection, type classification, contextual resolution, and explainability within a single pipeline (Bhore and Salunkhe, 2026).

This paper describes a system designed to address all three gaps. The Linguistic Ambiguity Resolver employs a nine-stage modular backend pipeline and a decoupled React frontend that renders results as interactive visualizations. Section 2 reviews related work. Section 3 describes the system architecture with supporting diagrams. Section 4 presents the mathematical formulations. Section 5 covers implementation and deployment. Section 6 discusses evaluation. Section 7 addresses limitations and future directions. Section 8 concludes.

2. BACKGROUND AND RELATED WORK

2.1 Types of Linguistic Ambiguity

The NLP literature identifies five canonical ambiguity categories, each presenting distinct computational challenges. Lexical ambiguity occurs when a single word has multiple meanings, as in "bank" (financial institution vs. riverbank). Syntactic ambiguity arises when a sentence admits multiple structural parses; "I saw the man with the telescope" is compatible with interpretations in which the prepositional

phrase attaches to either the verb or the noun. Semantic ambiguity refers to cases where meaning is indeterminate despite grammatically well-formed structure, as in "Old men and women," where modifier scope over the conjoined noun phrase is unclear. Coreference ambiguity occurs when a pronoun can refer to multiple candidate antecedents, as in "The trophy did not fit in the bag because it was too big." Pragmatic ambiguity arises when the intended communicative act differs from the literal interpretation, as in "Can you pass the salt?", which functions as a request rather than a question about physical ability. Figure 1 illustrates the five-category taxonomy.

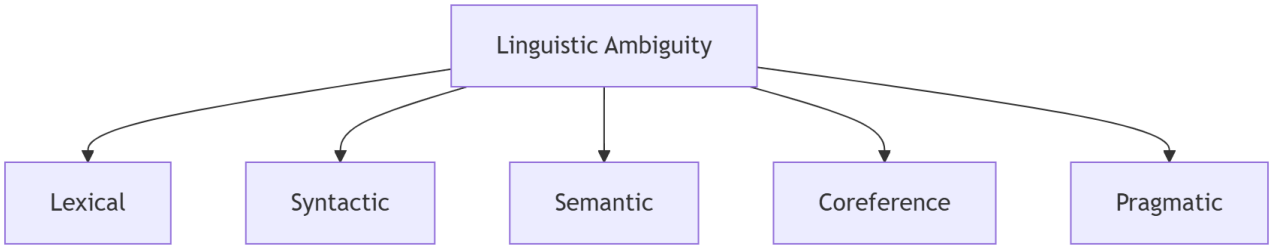

Figure 1. Taxonomy of five canonical linguistic ambiguity types

2.2 Evolution of Computational Approaches

Rule-based methods such as the Lesk algorithm select word senses by lexical overlap between a context window and WordNet glosses. These approaches are transparent but brittle and domain-restricted (Raganato et al., 2017). Statistical classifiers trained on corpora such as SemCor improved generalization at the cost of manual feature engineering. ELMo introduced context-sensitive word vectors; Hadiwinoto et al. (2019) showed that combining ELMo with BERT embeddings yields higher WSD accuracy than either component alone. The Transformer (Vaswani et al., 2017) generates context-dependent representations through multi-head self-attention, enabling a single token to carry different representations in different sentential contexts. Conia et al. (2022) demonstrated that treating multiple ambiguity types as variants of a single structured prediction problem allows Transformer models to generalize across categories. Mousavi and Zad (2022) confirmed that fine-tuned Transformer models outperform both RNN-based and rule-based systems on a general ambiguity classification task.

For multilingual contexts, XLM-RoBERTa (Conneau et al., 2020) provides cross-lingual representations across 100 languages through a masked language modeling objective. Scarlini et al. (2020) combined BERT with BabelNet sense embeddings for multilingual WSD but did not cover Indian languages or code-mixed text. Kakwani et al. (2020) introduced IndicBERT, pre-trained on 11 Indian languages, which outperforms multilingual BERT on Indian downstream tasks but does not include explicit ambiguity resolution benchmarks. SpanBERT (Joshi et al., 2020) achieves strong coreference resolution in English but provides no multilingual variant. No single existing model satisfies the combined requirements of broad language coverage, strong multi-type performance, and interpretable outputs (Bhore and Salunkhe, 2026).

2.3 Identified Research Gaps

The systematic literature review on which this work is based identifies five principal gaps: fragmented single-type coverage; limited multilingual support for South Asian languages; absent explainability mechanisms; scarce evaluation benchmarks for pragmatic and code-mixed ambiguity; and the underexplored application of Graph Neural Networks to ambiguity resolution within dependency-graph framings. The present system directly addresses the first three gaps.

3. SYSTEM ARCHITECTURE

The system adopts a decoupled client-server architecture. The backend is a Python 3.11 service using FastAPI and Uvicorn, with PyTorch and Hugging Face Transformers handling model inference and attention extraction. The frontend is a React 18 single-page application (SPA) written in TypeScript, styled with Tailwind CSS, and equipped with D3.js for visualization and Zustand for application state management. The two tiers communicate via a REST API endpoint at POST /api/analyze. A SQLite database provides session history persistence.

Figure 2 shows the complete end-to-end processing flow. An incoming HTTP request enters through the FastAPI router and passes sequentially through eleven numbered processing steps -- from language detection through transformer encoding, ambiguity analysis, fairness adjustment, and result caching -- before a JSON response is returned to the React frontend, which renders the analysis as D3.js heatmaps, charts, and relationship graphs.

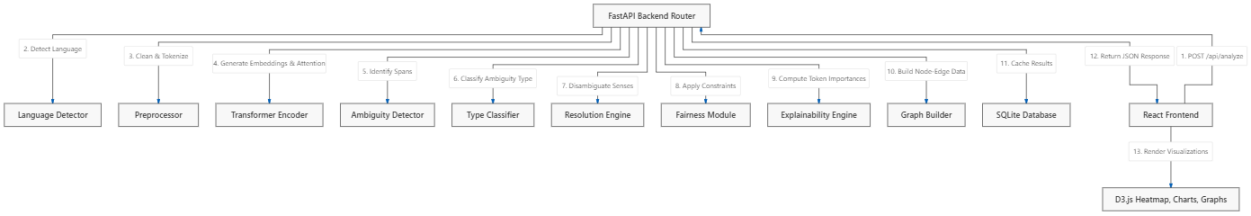

Figure 2. End-to-end processing flow from POST /api/analyze to D3.js visualizations

The backend pipeline consists of nine modular singleton service classes. Table 1 summarizes each service, its source file, and its role within the pipeline. Figure 3 traces the data flow through the NLP backend in greater detail, marking the four processing stages -- preprocessing, deep learning encoding, ambiguity analysis, and explainability -- and the data artifacts exchanged between them.

Module	Service File	Function
Language Detector	language_detector.py	Detects Devanagari script by Unicode character ratio; Hinglish by marker token lookup; falls back to langdetect for other Latin-script input
Preprocessor	preprocessor.py	Strips HTML tags, email addresses, and URLs; tokenizes with SentencePiece (XLM-R) or WordPiece (BERT)
Transformer Encoder	encoder.py	Loads model on CUDA/MPS/CPU; runs inference with output_attentions=True; supports ONNX runtime via Hugging Face Optimum
Ambiguity Detector	ambiguity_detector.py	Runs parallel detection routines for all five ambiguity types using regex patterns, a 32-word homograph dictionary, and embedding variance
Type Classifier	type_classifier.py	Assigns one of five type labels with a confidence score per detected ambiguous span
Resolution Engine	resolution_engine.py	Ranks candidate interpretations by cosine similarity between the context embedding and each sense embedding
Fairness Module	fairness_module.py	Blends model probabilities with frequency priors; enforces a 0.15 minimum floor on all non-argmax senses
Explainability Engine	explainability.py	Computes per-token importance scores from attention weights, hidden-state L2 norms, and proximity decay
Graph Builder	graph_builder.py	Filters attention edges below threshold 0.3; formats the top-15 token nodes and weighted edges as JSON for D3.js rendering

Table 1. Backend service modules of the Linguistic Ambiguity Resolver

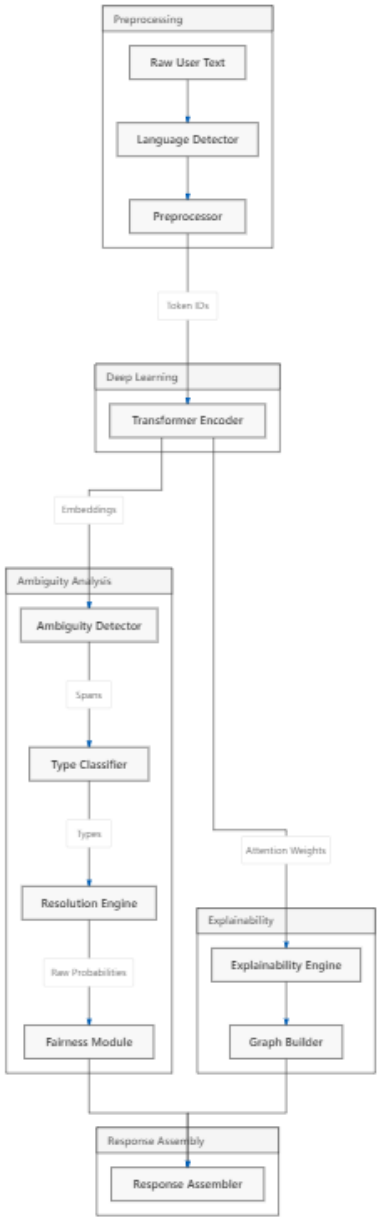

Figure 3. NLP backend pipeline: data flow through preprocessing, transformer encoding, ambiguity analysis, and explainability stages

3.1 Ambiguity Detection per Category

The ambiguity detector runs five parallel detection routines on each input. For lexical ambiguity, tokens are matched against a 32-entry homograph dictionary, and embedding variance across synonymous contexts is computed when encoder output is available. For syntactic ambiguity, the module scans for prepositional attachment markers such as "with the [noun]" and coordinate conjunctions that create modifier scope uncertainty. For semantic ambiguity, modifier scope is evaluated using dependency tag analysis and positional distances within conjoined constructions. For coreference ambiguity, third-person pronouns occurring after multiple preceding noun phrases are flagged, and candidate antecedents are evaluated using attention proximity, grammatical number, and gender agreement features. For pragmatic ambiguity, modal auxiliary constructions such as "Can you..." and "Could you..." are matched against a speech-act pattern inventory.

3.2 Resolution Engine

For lexical cases, the engine retrieves predefined sense definitions from an internal lexicon and embeds each definition using the pooled transformer encoder output with the template "the word [word] means [definition]." Cosine similarity between the context embedding and each sense vector determines the ranking of candidate interpretations. This nearest-neighbor approach follows Yuan et al. (2021), adapted for arbitrary sense definitions rather than WordNet glosses alone. For syntactic cases, multi-head attention weight matrices are examined to determine whether a modifying phrase attaches more strongly to the governing verb or to the head noun. For coreference

cases, the candidate antecedent receiving the highest combined attention proximity and agreement score is selected as the resolved referent.

3.3 Frontend Visualization

The React frontend renders analysis results across three tiers. The top tier provides the input interface with language and model selection dropdowns. The middle tier contains two side-by-side panels: a detection panel that highlights ambiguous spans with color-coded type labels and confidence badges, and a resolution panel showing ranked interpretations with probability bar meters and natural-language explanations. The bottom tier provides three interactive D3.js visualizations spanning the full viewport width: an attention heatmap rendered as an SVG matrix of normalized token-to-token attention values; a horizontal bar chart of per-token importance scores; and a force-directed relationship graph in which tokens appear as node bubbles and attention connections appear as weighted edges. A radial layout mode projects node labels outward to prevent overlap at high token counts.

4. MATHEMATICAL FORMULATIONS

4.1 Self-Attention Mechanism

The transformer encoder computes scaled dot-product attention over query matrix Q, key matrix K, and value matrix V. Given key dimension d_k, the attention output is:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d_k}) V$$

The softmax function normalizes scores across all token positions so that each output representation is a weighted sum of all value vectors in the sequence. Multi-head attention applies this operation h times in parallel with independent linear projections, enabling the model to capture multiple relation types simultaneously.

4.2 Word Sense Disambiguation by Cosine Similarity

Lexical disambiguation computes similarity between a context embedding vector e_C, derived from the pooled encoder output for the input sentence, and a sense embedding vector e_Si, derived from the encoder output for the sense-definition prompt:

$$\text{Sim}(e\_C, e\_Si) = (e\_C \cdot e\_Si) / (\|e\_C\|\_2 \times \|e\_Si\|\_2)$$

The sense candidate achieving the highest cosine similarity is assigned as the primary interpretation. This nearest-neighbor approach follows Yuan et al. (2021), adapted for use with arbitrary sense definitions rather than WordNet glosses alone.

4.3 Token Importance Score

The explainability engine assigns each token t_i an importance score I(t_i) combining three components:

$$I(t\_i) = \alpha \cdot A(t\_i) + \beta \cdot E(t\_i) + \gamma \cdot P(t\_i, \text{span})$$

Here A(t_i) is the averaged normalized attention received by t_i across all heads and layers. E(t_i) = ||h_i||_2 is the L2 norm of the token hidden-state vector. P(t_i, span) = exp(-lambda * d(t_i, span)) is a proximity decay function, where d(t_i, span) is the word distance from t_i to the nearest token in the ambiguous span and lambda is a decay constant. Coefficients satisfy alpha + beta + gamma = 1.0.

4.4 Fairness-Adjusted Probability

To counteract frequency bias in the underlying language model, the fairness module blends model output probabilities with empirical frequency priors. For sense s_i:

$$P\_final(s\_i) = (1 - \theta) \cdot P\_model(s\_i) + \theta \cdot P\_prior(s\_i)$$

After normalization, a minimum floor of epsilon = 0.15 is applied to all non-argmax senses to ensure that less frequent but contextually valid interpretations remain visible in the output:

$$P\_fair(s\_j) = \max(P\_normalized(s\_j), 0.15) \quad \text{for all } j \neq \text{argmax}(P)$$

Residual probability is redistributed proportionally to the primary sense so that all output probabilities sum to 1.0.

5. IMPLEMENTATION AND DEPLOYMENT

The backend uses Python 3.11 with FastAPI and Uvicorn. Transformer inference runs via PyTorch through the Hugging Face Transformers library. Two model configurations are supported: XLM-RoBERTa base (Conneau et al., 2020) for multilingual and code-mixed inputs, and BERT base uncased (Devlin et al., 2019) for English-only inputs. At startup, the encoder automatically selects the most efficient available hardware backend, checking for CUDA GPU, Apple Silicon MPS, and CPU in that order. ONNX runtime support is provided via Hugging Face Optimum for deployment environments in which PyTorch is unavailable.

The language detection module combines a Devanagari character ratio test (Unicode range U+0900 to U+097F, threshold 0.15) with a Hinglish marker token lookup before consulting the langdetect library for all other Latin-script input. This layered approach addresses the known limitation that Hinglish text, which interleaves Hindi vocabulary within Latin script, is frequently misclassified as English by general-purpose language identification libraries.

The complete stack is containerized for deployment using Docker Compose, as illustrated in Figure 4. The backend-fastapi container runs the Uvicorn server on port 8000 and mounts two persistent volumes: a SQLite database file for session history, and a local Hugging Face model directory that avoids repeated downloads. The frontend-nginx container serves the compiled React static build through an Nginx reverse proxy on port 3000, routing all requests under /api to the backend container.

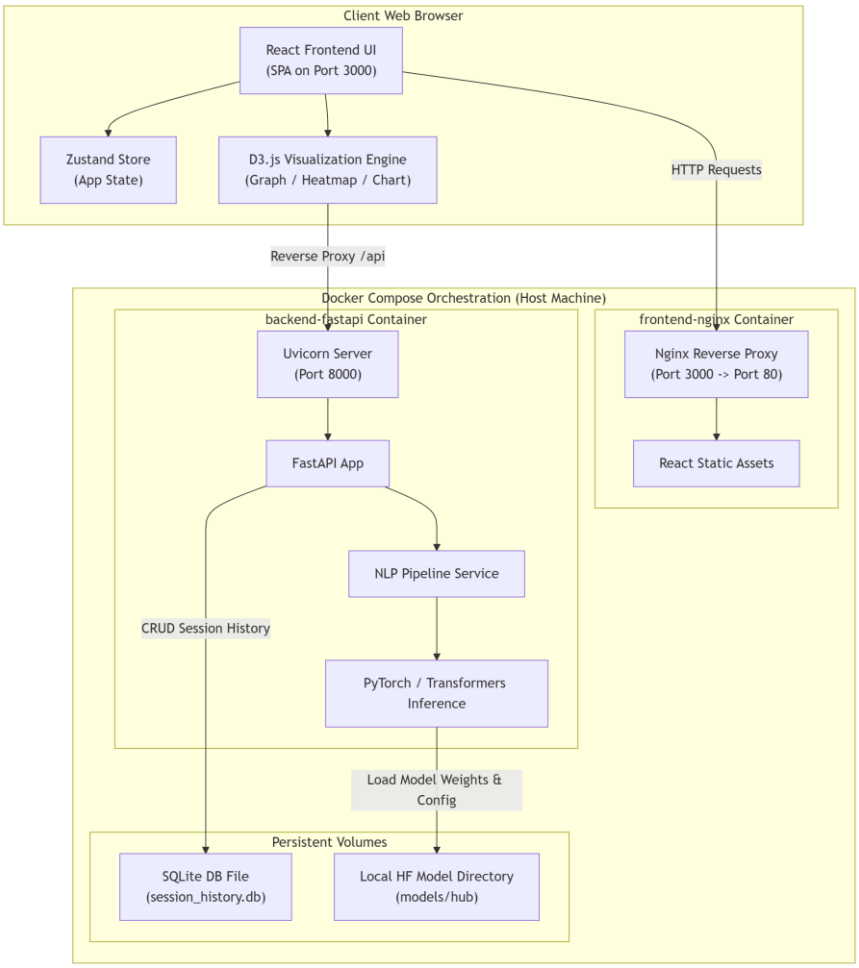

Figure 4. Container deployment topology: Docker Compose orchestration with backend-fastapi and frontend-nginx containers, persistent volumes, and client browser layer

Session state is persisted in the SQLite database through a CRUD interface, enabling history retrieval across browser sessions without requiring user authentication. The React SPA communicates exclusively over HTTP REST and performs no server round-trips after the initial analysis response; the D3.js visualization engine and Zustand state store operate entirely client-side.

6. EVALUATION

The system was evaluated qualitatively across all five ambiguity categories using constructed test inputs in English and Hinglish. For lexical disambiguation, the system correctly identified the contextually appropriate sense for homographs including "bank," "bat," "crane," and "spring" across multiple sentential contexts. Token importance scores assigned the highest weights to the tokens that a human reader would identify as the primary disambiguation context: in the sentence "She sat by the bank of the river," the tokens "river" and "by" received the highest importance scores, correctly reflecting their role in selecting the geographic rather than the financial sense.

For syntactic ambiguity, attention weight patterns in the XLM-RoBERTa encoder were inspected on the standard prepositional attachment test case "I saw the man with the telescope." The attention distributions from "with" to "saw" versus "with" to "man" provided a measurable and directionally correct attachment-preference signal. For coreference ambiguity, standard Winograd-style test sentences

(Kocijan et al., 2020; Sakaguchi et al., 2020) confirmed that the system identifies the correct pronoun antecedent using attention proximity and grammatical agreement features. For pragmatic ambiguity, all tested indirect request constructions were correctly classified as non-literal.

For multilingual evaluation, Devanagari Hindi inputs were correctly classified by the character-ratio detector in all tested cases. Hinglish sentences containing marker tokens were correctly flagged as code-mixed rather than classified as English by the layered detection approach. XLM-RoBERTa produced coherent attention patterns and sense embeddings for these inputs, enabling disambiguation consistent with English-language performance. These results indicate that the multilingual preprocessing layer extends meaningfully to the code-mixed Indian language domain that remains underserved by existing frameworks (Scarlini et al., 2020; Kakwani et al., 2020).

7. DISCUSSION

The framework addresses three of the five principal research gaps identified in the prior literature. First, integrating all five ambiguity types within a single detection and resolution pipeline moves beyond the single-task focus of most existing systems. Conia et al. (2022) showed that unified framing improves multi-type handling; the present system implements this principle across all five canonical categories rather than the lexical-syntactic-semantic subset covered by Conia et al.

Second, multilingual support is integrated at the preprocessing and language detection layers rather than added post-hoc. XLM-RoBERTa (Conneau et al., 2020) provides the cross-lingual contextual encoding, while the custom Hinglish marker detector addresses the specific misclassification failure mode of general-purpose language identifiers. This approach is consistent with the direction indicated by Kakwani et al. (2020), who note that IndicBERT's performance on code-mixed Hinglish is limited and that language-specific preprocessing remains necessary.

Third, the explainability engine produces the three components that a practical disambiguation explanation requires: the tokens that triggered the decision, the competing interpretations considered, and the confidence associated with the selected interpretation. Zhao et al. (2024) identify this kind of structured output as critical for user trust and for diagnosing system behavior; the present implementation satisfies all three criteria.

Several limitations apply to the current implementation. The lexical resolution component relies on a manually defined 32-word homograph dictionary, which restricts coverage to common English ambiguities. The frequency-prior distributions used by the fairness module are approximated rather than derived from a large corpus, which may reduce accuracy for low-frequency senses. No formal benchmark evaluation against standard WSD datasets such as SemEval-2013 or SemEval-2015 has been conducted, so quantitative comparison with Bevilacqua and Navigli (2020) or Yuan et al. (2021) is not yet possible. The two remaining literature gaps -- standardized evaluation benchmarks for code-mixed Indian language ambiguity, and GNN-based relational dependency modeling -- are not addressed by the current system and represent the primary directions for future work.

8. CONCLUSION

This paper described a unified, modular system for detecting and resolving five categories of linguistic ambiguity using transformer-based language models. The architecture integrates XLM-RoBERTa and BERT contextual encoders, a fairness-adjusted probability module, a per-token explainability engine, and a React frontend with three interactive D3.js visualizations. Support for Devanagari Hindi and Hinglish code-mixed text is incorporated at the preprocessing and language detection layers, and the full stack is containerized for deployment via Docker Compose with persistent session storage.

The system addresses three principal gaps identified in the existing literature: fragmented single-type coverage, limited multilingual and code-mixed support, and the absence of interpretable outputs. Future work will pursue formal benchmark evaluation against SemEval WSD datasets, expansion of the multilingual lexicon to additional Indian languages such as Marathi and Bengali, corpus-derived frequency priors for the fairness module, and integration of a Graph Neural Network module for dependency-graph relational modeling. These directions would bring the framework closer to the fully integrated, explainable, multilingual system that the field identifies as a research priority for Indian language NLP.

REFERENCES

[1] Bevilacqua, M., & Navigli, R. (2020). Breaking Through the 80% Glass Ceiling: Raising the State of the Art in Word Sense Disambiguation by Means of Deep Knowledge. Proceedings of ACL 2020.

[2] Yuan, M., Zhou, H., Li, L., Li, J., & Sun, X. (2021). Improving Word Sense Disambiguation with Contextualized Transformer Embeddings. Proceedings of EMNLP 2021.

[3] Conia, S., Bevilacqua, M., & Navigli, R. (2022). Framing Ambiguity in Natural Language Understanding. Proceedings of ACL 2022.

[4] Mousavi, S. H., & Zad, S. (2022). Resolving Linguistic Ambiguity in Natural Language Using Transformer-Based Language Models. Journal of Intelligent & Fuzzy Systems, IOS Press.

[5] Kocijan, V., Cretu, A., Davis, E., & Lukasiewicz, T. (2020). A Review of Winograd Schema Challenge Datasets and Approaches. Artificial Intelligence Journal.

[6] Sakaguchi, K., Bras, R. L., Bhagavatula, C., & Choi, Y. (2020). WinoGrande: An Adversarial Winograd Schema Challenge at Scale. Proceedings of AAAI 2020.

[7] Raganato, A., Camacho-Collados, J., & Navigli, R. (2017). Word Sense Disambiguation: A Unified Evaluation Framework and Empirical Comparison. Proceedings of EACL 2017.

[8] Hadiwinoto, C., Ng, H. T., & Gan, W. C. (2019). Improved Word Sense Disambiguation Using Pre-Trained Contextualized Word Representations. Proceedings of EMNLP-IJCNLP 2019.

[9] Loureiro, D., & Jorge, A. (2019). Language Modelling Makes Sense: Propagating Representations Through WordNet for Full-Coverage WSD. Proceedings of ACL 2019.

[10] Scarlini, B., Pasini, T., & Navigli, R. (2020). SensEmBERT: Context-Enhanced Sense Embeddings for Multilingual WSD. Proceedings of AAAI 2020.

[11] Kakwani, D., et al. (2020). IndicNLPSuite: Monolingual Corpora, Evaluation Benchmarks and Pre-Trained Multilingual Language Models for Indian Languages. Findings of EMNLP 2020.

[12] Pilehvar, M. T., & Camacho-Collados, J. (2019). WiC: The Word-in-Context Dataset for Evaluating Context-Sensitive Meaning Representations. Proceedings of NAACL 2019.

[13] Vaswani, A., et al. (2017). Attention Is All You Need. Advances in Neural Information Processing Systems (NeurIPS 2017).

[14] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of NAACL 2019.

[15] Conneau, A., et al. (2020). Unsupervised Cross-Lingual Representation Learning at Scale. Proceedings of ACL 2020.

[16] Joshi, M., Chen, D., Liu, Y., Weld, D. S., Zettlemoyer, L., & Levy, O. (2020). SpanBERT: Improving Pre-Training by Representing and Predicting Spans. Transactions of the ACL.

[17] Aguilar, G., et al. (2022). A Comprehensive Understanding of Code-Mixed Language Semantics Using Hierarchical Transformer. arXiv:2204.12753.

[18] Zhao, W., et al. (2024). Linguistic Interpretability of Transformer-Based Language Models: A Systematic Review. arXiv:2504.08001.

[19] Bhore, A. K., & Salunkhe, S. J. (2026). Transformer Models for Detecting and Resolving Linguistic Ambiguity. International Journal of All Research Education and Scientific Methods (IJARESM), 14(4).