# Transformer Encoders with Heuristic-Guided Contrastive Learning for Software Coreference Resolution

**Mahmoud Hassan**[1] **Dipendra Yadav**[2]
[1]ZBW - Leibniz Information Centre for Economics
[2]University of Greifswald
[1]m.hassan@zbw.eu, [2]dipendra.yadav@uni-greifswald.de

**Abstract**

This paper describes our system submitted to the Software Mention Detection and Coreference Resolution (SOMD) 2026 shared task, specifically for Subtask 1 (cross-document coreference resolution over gold-standard mentions) and Subtask 2 (cross-document coreference resolution over predicted mentions). The proposed approach employs a SciBERT architecture trained with Supervised Contrastive (SupCon) loss to generate dense mention representations, which are then clustered using Hierarchical Agglomerative Clustering (HAC) with average linkage. Software-aware heuristics are integrated to exploit domain-specific signals such as software name canonicalization and developer disambiguation to adjust pairwise similarity scores before clustering. The system achieved strong performance, with a CoNLL F1 score of 92.18% on coreference resolution over gold-standard mentions and 91.87% on coreference resolution over predicted mentions, showing significant performance of our approach in this area for human annotated and automated systems respectively.

**Keywords:** Coreference resolution, Software mention, BERT, Clustering, Contrastive Learning

## 1. Introduction

Software plays a critical role in scientific research, yet it remains one of the most inconsistently cited entities in scholarly publications (Schindler et al., 2024). The same software tool may be referenced under different names, versions, or abbreviations in multiple documents, making cross-document coreference resolution (CDCR) an essential task for building comprehensive research knowledge graphs. the shared task Software Mention Detection and Coreference Resolution (SOMD) 2026 addresses this challenge by focusing on identifying and clustering software mentions in scientific documents.

This paper contributes to two subtasks: Subtask 1 concerns coreference resolution over gold-standard mentions, where mention boundaries and types are provided, and the system must cluster coreferent mentions. Subtask 2 addresses coreference resolution over predicted mentions, where the system must handle noisier input from an upstream mention detection module. Both subtasks require grouping all mentions that refer to the same software entity into coreference clusters. Our approach focuses on a robust, scalable architecture that balances neural semantic similarity with domain-specific logic.

The system is built around three core components: (1) a SciBERT encoder (Beltagy et al., 2019) trained with supervised contrastive loss (Khosla et al., 2020) to produce cluster-friendly embeddings; (2) a set of software-aware heuristics that leverage domain-specific signals such as software name canonicalization, version numbers, and developer metadata to adjust pairwise similarity scores; and (3) Hierarchical Agglomerative Clustering (HAC) (Manning et al., 2008) for final cluster formation. This paper details the proposed approach, discusses the rationale behind each design decision with illustrative examples, and presents the experimental results.

All relevant code and implementations have been made publicly available on GitHub[1] for reproducibility and further research.

## 2. Related Work

Coreference resolution involves grouping textual expressions (mentions) referring to the same real-world entity into clusters (Pradhan et al., 2012). Although intra-document coreference has seen significant advances with Transformer-based models (Joshi et al., 2020), Cross-Document Coreference Resolution (CDCR) presents a wider search space and increased ambiguity (Cattan et al., 2021).

CDCR systems typically follow either a *Cross-Encoder* architecture, which models token-level interactions between mention pairs at $O(N^2)$ cost, or a *Bi-Encoder* (or Siamese) architecture, which encodes mentions independently into a shared semantic space (Humeau et al., 2020). The latter is preferred for large-scale tasks such as SOMD Subtasks 2 and 3 due to its efficiency in high-volume clustering.

The SoMeSci corpus (Schindler et al., 2021) provides gold-standard annotations for software mentions in scientific articles, including mention types (e.g., usage, creation), software types (e.g.,

[1]https://github.com/MahmoudMHassan/SOMD-2026



*Proceedings of the 3rd Int. Workshop on Natural Scientific Language Processing (NSLP 2026) @ LREC 2026*, pages 270–276
12 May 2026. 

application, plugin), and relational metadata such as version numbers and developers. The SOMD shared task series (2024 (Krüger et al., 2024), 2025 (Upadhyaya et al., 2025), 2026) builds on this foundation, progressively introducing more challenging subtasks, including mention detection, relation extraction, and cross-document coreference resolution.

### 2.1. Contrastive Learning for Clustering

Metric learning aims to learn an embedding space where similar items are proximity-wise closer than dissimilar ones. A fundamental objective for this is triplet loss, popularized by FaceNet for face recognition (Schroff et al., 2015). Triplet loss operates on anchor-positive-negative triplets, ensuring that the anchor is closer to the positive by a specific margin.

However, Contrastive learning has emerged as a powerful paradigm for learning representations suitable for clustering tasks. The InfoNCE loss (van den Oord et al., 2018) and its supervised extension, Supervised Contrastive (SupCon) Loss (Khosla et al., 2020), provides a superior representation density for clustering and training models to produce embeddings where the same-class instances are pulled together and different-class instances are pushed apart. Unlike triplet loss, which focuses on a single positive/negative pair per anchor in a "local" update, SupCon contrasts an anchor against all other positives and negatives in a batch simultaneously. This "all-pairs" optimization results in more compact clusters and better separation, making it ideal for the CDCR objective, where the downstream task is direct clustering of the learned representations and the final evaluation is based on cluster metrics like CoNLL F1.

### 2.2. Data and task description

Given a mention, its type, and relations, the task is to form clusters of mentions (mention ids) referring to the same software. There are three subtasks in the SOMD 2026 challenge, where the difference between the three subtasks lies in the data source and size. For subtask 1, software mentions, types, and relations were human-annotated. For subtask 2, the mentions, types, and relations were automatically predicted by the entity and relation extraction models. The goal is to assess how well the system performs with noisy predictions. For Subtask 3, the data set is also predicted using a baseline model; however, the number of entities is large, and the evaluation would also include the runtime test. This paper describes the approach used in subtask1 and subtask2. Subtask 1 provides gold-standard mentions: 2,974 training mentions in 733 clusters and 743 test mentions. Subtask 2 provides predicted mentions: 2,860 training mentions in 699 clusters and 12,516 test mentions. Each mention includes the surface form, type, document identifier, sentence context, and relational metadata (version, developer, abbreviation).

### 2.3. Evaluation Metrics

All submissions are evaluated using standard coreference resolution metrics. The metrics are Message Understanding Conference (MUC (Vilain et al., 1995)) which measures how well predicted clusters recover gold coreference links, The B-Cubed metric ($B^3$ (Bagga and Baldwin, 1998)) which computes mention-level precision and recall based on cluster overlap and the Constrained Entity-Aligned F-measure (CEAFe (Luo, 2005)) which measures the similarity between predicted and gold clusters via optimal alignment. Lastly, the CoNLL Metric ((Pradhan et al., 2012)) is defined as the unweighted macro-average of the F1-scores of MUC, $B^3$, and CEAFe which is their arithmetic mean. The challenge results were based exclusively on the CoNLL F1-score, and all metrics are computed on the full test set and reported at the cluster level.

## 3. System description

The system implements a three-stage pipeline: (1) mention encoding using a SciBERT model, (2) heuristic-guided pairwise similarity adjustment, and (3) Hierarchical Agglomerative Clustering for final cluster formation. The same pipeline and trained model are used for both Subtask 1 and Subtask 2.

### 3.1. Mention Representation

Each mention is represented as a structured text string that combines the mention surface form with its surrounding sentence context. For each mention $m$, we construct the input as:

`"mention_text [SEP] sentence_context"`

where `mention_text` is the software name (e.g., "MATLAB") and `sentence_context` is the sentence containing the mention. This dual-input format, separated by the `[SEP]` token, is a standard practice in BERT-based entity representation tasks (Soares et al., 2019). It allows the self-attention mechanism to model the interaction between the entity name and its specific usage context, which is critical for disambiguating mentions like "Python" (the language) versus "Python" (the snake).

For example, given the mention "NQuery Advisor" appearing in the sentence "NQuery Advisor by StatsSols version 3.0 was used to calculate sample size", the input to the encoder becomes:

"NQuery Advisor [SEP] NQuery Advisor by StatsSols version 3.0 was used to calculate sample size".

This format exploits BERT's [SEP] token to delineate the mention identity from its context, allowing the model to attend to both components. The surface form of mention appearing twice—once isolated and once in context—provides the encoder with complementary views of the entity.

### 3.2. Model Architecture

A transformer based approach with shared weights is built on SciBERT[2] (Beltagy et al., 2019), a BERT-based model pre-trained on 1.14 million scientific papers. Its domain-specific vocabulary (scivocab) provides better subword tokenization for software names than general-purpose BERT, reducing information loss from excessive subword fragmentation. The architecture consists of three components: (1) a SciBERT encoder producing 768-dimensional token representations; (2) a Pooling Layer that extracts the [CLS] token as the aggregate sequence embedding; and (3) a Projection Head—a two-layer MLP (Linear$(768, 512) \rightarrow$ ReLU $\rightarrow$ Linear$(512, 256)$)—that maps the embedding to a 256-dimensional contrastive space. As noted by (Chen et al., 2020), this non-linear projection preserves the pre-trained features during contrastive training. Each mention is encoded independently in $O(n)$ passes, with pairwise similarities computed over cached embeddings, enabling efficient scaling to the 12,516 mentions in Subtask 2.

### 3.3. Supervised Contrastive Loss

The model is trained with supervised contrastive loss (SupCon) (Khosla et al., 2020). Unlike Triplet Loss (Schroff et al., 2015), which requires expensive semi-hard negative mining and optimizes one anchor–positive–negative triplet at a time, SupCon leverages the full label structure in each batch: for an anchor mention $A$ in cluster $C$, all other mentions from $C$ are treated as positives and all others as negatives simultaneously.

Consider a batch with three mentions of "MATLAB" (cluster A) and two of "SPSS" (cluster B). In a single step, SupCon (1) pulls all MATLAB mentions together, (2) pulls both SPSS mentions together, and (3) pushes all MATLAB mentions away from all SPSS mentions. This "all-pairs" global optimization produces denser intra-cluster and clearer inter-cluster separation than triplet loss, which is precisely the structure required for downstream clustering evaluated by CoNLL F1.

[2] https://huggingface.co/allenai/scibert_scivocab_uncased

The loss uses a temperature $\tau = 0.07$ (Chen et al., 2020), which sharpens decision boundaries and increases sensitivity to subtle differences between non-coreferent mentions (e.g., "MATLAB" vs. "MATLAB R2019b").

### 3.4. Software-Aware Heuristics

Software coreference resolution carries strong domain-specific signals that pure neural models may not fully exploit due to the lexical and semantic sparsity of software mentions. We introduce a heuristic layer that modifies the pairwise cosine distance matrix before clustering, framing this as a specialized form of domain-specific Entity Resolution (ER) (Čuljak et al., 2022).

Concretely, we apply two heuristics to adjust the similarity score $s(m_i, m_j)$ between any two mentions $m_i$ and $m_j$ of distinct documents:

**Name Canonicalization** We compute a canonical form $C(m)$ (lowercase, punctuation removal, abbreviation expansion). If $C(m_i) = C(m_j)$, we apply a boost $\delta_{\text{name}} = 0.5$ to the similarity score.

**Developer Disambiguation** Conflicting developer metadata (e.g. "Adobe" vs "Microsoft") incurs a penalty $\delta_{\text{dev}} = -0.8$, acting as a strong disambiguator for semantically similar entities (e.g., different PDF readers). This disambiguates semantically similar entities, such as "PyTorch" and "TensorFlow", which share nearly identical contextual embeddings (as deep learning frameworks) but are distinct software tools.

Each adjustment is applied additively to the cosine similarity matrix. Delta values ($\delta_{\text{name}} = 0.5$, $\delta_{\text{dev}} = -0.8$) were determined through a grid search in the validation split, sweeping both parameters independently over the range $[-1.0, +1.0]$ in increments of 0.1, and selecting the combination that maximized CoNLL F1 on held-out clusters. The large magnitude of $\delta_{\text{dev}}$ reflects the empirical finding that conflicting developer metadata is a near-definitive signal of non-coreference, while the moderate positive value of $\delta_{\text{name}}$ reflects that canonical name equality is strong but not infallible evidence of coreference (e.g., two distinct tools may share a generic name).

### 3.5. Hierarchical Agglomerative Clustering

Given the heuristic-adjusted similarity matrix $S'$, we apply Hierarchical Agglomerative Clustering (HAC) to produce the final clusters of coreference. We define the distance between two mentions as $D'_{ij} = 1 - S'_{ij}$. HAC iteratively merges clusters $C_a$ and $C_b$ based on the average linkage criterion, which

minimizes the average distance between all pairs of mentions across the two candidate clusters:

$$D(C_a, C_b) = \frac{1}{|C_a||C_b|} \sum_{m_i \in C_a} \sum_{m_j \in C_b} D'_{ij} \quad (1)$$

The choice of average linkage over alternatives like Connected Components (which is functionally equivalent to single-linkage clustering (SLC) ) or absolute complete linkage is motivated by robustness. Single-linkage methods are highly susceptible to the "chaining effect" (Čuljak et al., 2022), where a single noisy pair-wise prediction can erroneously merge two semantically distinct software clusters (semantic drift). Average linkage requires a consensus between all mention pairs, ensuring that the merged cluster remains coherent.

A fixed distance threshold $\theta = 0.5$ was used as the stopping criterion. This approach is naturally suited to CDCR as it does not require a-priori knowledge of the number of clusters $K$. The threshold was tuned in the validation set to maximize CoNLL F1.

### 3.6. Training Configuration

In CDCR, the data splitting strategy is critical to prevent data leakage and ensure the model learns generalized entity representations. We employ a Cluster-Level (Equivalence-Class) Splitting strategy: rather than splitting mentions independently, we assign entire coreference clusters to either the training or validation set. This ensures that if a software name (e.g., "SPSS") appears in the training set, it is completely absent from the validation set.

This strategy prevents the model from achieving high performance simply by memorizing specific software names; instead, it must learn to identify coreference based on context and surface-form patterns that generalize to unseen software entities.

The model is trained using the AdamW optimizer with a learning rate of $2 \times 10^{-5}$ and a batch size of 16 for 30 epochs. We implement early stopping with a patience of 4, monitored using CoNLL F1 validation. For Subtask 1, early stopping triggered at epoch 10. For Subtask 2, early stopping triggered at epoch 5. All experiments were conducted on a single Google Colab T4 GPU environment.

## 4. Results

For Subtask 1, our system produces 261 clusters from 743 test mentions. For Subtask 2, the same model and pipeline are applied to 12,516 test mentions, resulting in 2641 clusters. The evaluation results could be shown in Table 1 and Table 2 for the tasks, respectively.

| Metric | Precision | Recall | F1-Score |
|---|---|---|---|
| $B^3$ | 0.96 | 0.92 | 0.94 |
| CEAFe | 0.80 | 0.93 | 0.86 |
| MUC | 0.99 | 0.95 | 0.97 |
| **CoNLL F1** | **0.92** | | |

Table 1: Evaluation results on the test phase of subtask 1

| Metric | Precision | Recall | F1-Score |
|---|---|---|---|
| $B^3$ | 0.97 | 0.89 | 0.93 |
| CEAFe | 0.78 | 0.94 | 0.85 |
| MUC | 0.99 | 0.96 | 0.98 |
| **CoNLL F1** | **0.92** | | |

Table 2: Evaluation results on the test phase of subtask 2

### 4.1. Comparative Analysis of Model Variants

To evaluate the impact of our architectural choices, we compared our best system (Bi-Encoder with SupCon, Heuristics, and HAC) with three experimental variants explored in our preliminary runs as shown in Table 3.

| Setup | CoNLL F1 |
|---|---|
| Bi-Encoder + InfoNCE | 0.79 |
| Bi-Encoder + HNM | 0.81 |
| Cross-Encoder Re-ranking | 0.84 |
| **Bi-Encoder + SupCon + Heuristics + HAC** | **0.92** |

Table 3: Comparison of SciBERT-based model variants on the Test phase of Subtask 1.

The first variant serves as a baseline, utilizing a SciBERT architecture trained with standard InfoNCE loss and clustered via graph-based connected components (Single-Linkage Clustering). The second variant incorporates periodic hard negative mining (HNM) during training to improve cluster separation in the embedding space. The third variant employs a Cross-Encoder re-ranking stage to refine the clusters produced by the Bi-Encoder; however, this approach introduces a $O(N^2)$ complexity at the re-ranking phase, which is problematic for large-scale tasks like Subtask 2. Our best system, which integrates Supervised Contrastive (SupCon) loss and metadata-aware heuristics with Hierarchical Agglomerative Clustering (HAC) achieves better performance while maintaining linear scaling for encoding, which (Humeau et al., 2020) identify as a critical requirement for production CDCR systems.

**Mention level vs Cluster level splitting** The impact of using mention level splitting was observed in the beginning of the experimentation which caused data leakage where the model "sees" the same entity in both training and validation sets. During validation, it merely recognizes an entity it has already optimized for, rather than learning the generalized property of coreferent software mentions. It also caused cluster fragmentation which breaks the "equivalence class" nature of CDCR. Validation metrics such as CoNLL F1 were reduced because the gold standard for validation only contains a subset of each cluster, making it impossible to achieve perfect recall. In contrast to cluster level splitting which ensured zero leakage, where validation tests the model on software entities it has never seen. This simulates the real-world scenario (Subtask 2, 3) where the model must resolve the mentions for arbitrary software. It also ensured cluster integrity where Gold clusters in the validation set remain complete. The scoring metrics then reflect the true ability of the system to group the mentions.

**SupCon vs. InfoNCEloss** Following (Khosla et al., 2020), we observe that the supervised contrastive loss with HAC is more robust than InfoNCEloss with single-linkage clustering (SLC) as discussed in subsection 3.3 for CDCR. SupCon's global batch optimization prevents the "local minima" problems, where a model may satisfy the margin for a specific cluster while ignoring broader cluster dynamics.

## 4.2. The Role of Heuristics

Software Heuristics serve as a critical domain-specific refinement layer that complements the raw semantic power of SciBERT embeddings. These heuristics—which include name canonicalization, developer conflict detection, and exact name matching—provide a "safety net" that raw embeddings often lack. Their primary value over the variants presented in section 4.1 is the ability to enforce hard constraints and logical penalties based on metadata. For instance, the "Developer Conflict" heuristic prevents merging mentions that are semantically similar but belong to different developers, effectively reducing false positives in the clustering process. By adjusting the distance matrix before Hierarchical Agglomerative Clustering (HAC), these heuristics ensure that the final clusters respect known software attributes that deep learning models might otherwise overlook. To isolate the contribution of software-aware heuristics, we evaluate our system with and without the heuristic adjustment layer on the validation split as shown in Table 4. It was observed that the name canonicalization step specifically made a significant improvement to the final output. One of the main differences in the setup of the first 3 variants shown in Table 3, that it embeds the metadata (type, relations) that come with the training data directly into the SciBERT input string ex. `[MENTION] name [TYPE] type [REL] relations [SEP] sentence` , which may contain version and developer information directly in the input string, the SciBERT model learns to bake the heuristics into the embeddings themselves during training, allowing the model to learn the heuristics. In contrast, to our best system which used a simpler input as discussed in section 3.1 and relies on the aforementioned external "boost/penalty" adjustments.

| Configuration | CoNLL F1 |
|---|---|
| Bi-Encoder + SupCon + HAC (no heuristics) | 0.41 |
| + Developer Disambiguation ($\delta_{\text{dev}}$=-0.8) | 0.46 |
| + Name Canonicalization ($\delta_{\text{name}}$=+0.5) | 0.94 |

Table 4: Incremental impact of software-aware heuristics on CoNLL F1 (validation split). Each row adds one heuristic to the previous configuration.

## 4.3. Qualitative Analysis

To illustrate the behavior of the system, we present representative cluster predictions and common error patterns from Subtask 1.

**Successful Clustering** The system correctly groups mentions of "SPSS" appearing in diverse contexts—"SPSS v20," "SPSS Statistics," and "SPSS (IBM Corp)"—into a single cluster. The name canonicalization heuristic ($\delta_{\text{name}}$) increases pairwise similarity for these surface variants, while the SupCon-trained embeddings capture shared contextual patterns (e.g., "statistical analysis was performed using . . . ").

**Developer Disambiguation Success** The system correctly separates "Acrobat Reader" (Adobe) from "Foxit Reader" (Foxit Software), despite both sharing the contextual phrase "PDF reader." The developer disambiguation penalty ($\delta_{\text{dev}} = -0.8$) pushes these otherwise similar embeddings apart, preventing an erroneous merge.

**Typical Error: Rare Software** The most common errors occur with rare or domain-specific software tools that appear only once or twice in the corpus (singletons). Without sufficient training signal, the encoder produces generic embeddings for

such mentions, occasionally merging them with semantically similar but distinct tools. For example, a niche bioinformatics pipeline may be incorrectly clustered with a more common tool mentioned in a similar analytical context.

## 5. Conclusion

We presented a pipeline for cross-document software coreference resolution, combining a SciBERT bi-encoder trained with Supervised Contrastive (SupCon) loss, software-aware heuristics, and Hierarchical Agglomerative Clustering, achieving a CoNLL F1 of 92.18% and 91.87% on Subtask 1 and Subtask 2 respectively using an identical model across both. Our analysis highlights three core findings: SupCon's all-pairs optimization is directly aligned with cluster-based evaluation, making it particularly effective for CDCR; software-aware heuristics and neural representations are complementary, with symbolic constraints enforcing boundaries that embeddings alone cannot reliably capture; and cluster-level splitting is essential to ensure genuine generalization to unseen software entities.

### 5.1. Limitations and Future Work

Although the current system achieves high performance, our analysis of the optimized implementation reveals some runtime and technical constraints that offer opportunities for future research.

#### 5.1.1. Limitations

**Heuristic Brittleness** Our software-aware heuristics rely heavily on regular expressions and fixed string matching. While effective for common tools like "MATLAB" or "SPSS," this approach is brittle when encountering non-standard software naming conventions or complex, nested version strings (e.g., "v.2.1-beta.rev4"). Furthermore, the heuristics depend on the quality of relation extraction from the previous pipeline stage; errors in developer or version detection propagate directly into the coreference scoring.

**Runtime and Scalability** The proposed pipeline exhibits three distinct asymptotic stages: (i) SciBERT encoding is $O(n)$, producing one embedding per mention in a single forward pass; (ii) heuristic adjustment is $O(n^2)$, iterating over all mention pairs to apply name canonicalization and developer-metadata comparisons; and (iii) HAC has a worst-case complexity of $O(n^3)$, reduced to $O(n^2 \log n)$ in practice via SciPy's Lance–Williams update formula. Crucially, the quadratic and super-linear stages operate over lightweight precomputed scalar distances, not neural forward passes, which justifies their inclusion relative to a Cross-Encoder baseline that would require full neural forward passes $O(n^2)$. Nevertheless, both the heuristic and the HAC stages require materializing the entire $n \times n$ similarity matrix. This quadratic memory and time footprint make the current pipeline infeasible as-is for datasets substantially larger than Subtask 2, such as Subtask 3.

**Global Clustering Threshold** We employ a fixed global threshold ($\theta = 0.5$) for HAC. This assumes that the optimal merging distance is consistent across all software domains. In practice, mature software ecosystems with many sub-modules (e.g., "Apache" projects) may exhibit different semantic densities than monolithic tools, potentially requiring adaptive or cluster-specific thresholds.

#### 5.1.2. Future Work

**Knowledge Graph Grounding** Beyond lexical heuristics, grounding software mentions in external Knowledge Graphs (KGs) such as WikiData or specialized software catalogs (e.g. SoftwareKG) would provide a robust disambiguation signal. Symbolic relations from a KG (e.g., *owned_by*, *successor_of*) could replace manual developer rules, allowing the system to resolve aliases that are semantically distinct but referentially identical.

**Approximate Nearest Neighbors and Poly-encoders** To address scalability, we plan to integrate FAISS-based approximate nearest neighbor search to narrow the clustering search space. Additionally, replacing the Bi-Encoder with a Poly-encoder (Humeau et al., 2020) could provide the fine-grained token interaction of a Cross-Encoder while maintaining the inference speed required for large-scale CDCR.

**Adaptive Threshold Learning** We intend to replace the fixed threshold with a learned similarity metric or a meta-learning approach that adjusts merging criteria based on the local density of the embedding space, improving performance in highly fragmented software clusters.

## 6. Acknowledgments

The authors thank the organizers of the Shared task on Software Mention Detection and Coreference Resolution (SOMD) and the Natural Scientific Language Processing (NSLP) workshop for running this competition. We also thank the reviewers for their insightful and constructive comments, which helped raise the standard of this manuscript considerably.

## 7. Bibliographical References


Amit Bagga and Breck Baldwin. 1998. Algorithms for scoring coreference chains. In *The first international conference on language resources and evaluation workshop on linguistics coreference*, volume 1, pages 563–566.

I. Beltagy, K. Lo, and A. Cohan. 2019. Scibert: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3615–3620, Hong Kong, China. Association for Computational Linguistics.

A. Cattan, A. Eirew, G. Stanovsky, M. Joshi, and I. Dagan. 2021. Cross-document coreference resolution over predicted mentions. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 5100–5107.

T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. 2020. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, pages 1597–1607.

I. Čuljak, A. Spitz, R. West, and A. Arora. 2022. Strong heuristics for named entity linking. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1826–1845.

S. Humeau, K. Shuster, M. Lachaux, and J. Weston. 2020. Poly-encoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*.

M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy. 2020. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan. 2020. Supervised contrastive learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 18661–18673.

Frank Krüger, Saurav Karmakar, and Stefan Dietze. 2024. SOMD@NSLP2024: Overview and insights from the software mention detection shared task. In *Natural Scientific Language Processing and Research Knowledge Graphs (NSLP 2024)*, pages 216–233, Cham. Springer Nature Switzerland.

X. Luo. 2005. On coreference resolution performance metrics. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP 2005)*, pages 25–32.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

S. Pradhan, A. Moschitti, N. Xue, O. Uryupina, and Y. Zhang. 2012. CoNLL-2012 shared task: Modeling multilingual unrestricted coreference in OntoNotes. In *Proceedings of the Joint Conference on EMNLP and CoNLL - Shared Task*, pages 1–40, Jeju Island, Korea.

D. Schindler, F. Bensmann, S. Dietze, and F. Krüger. 2021. Somesci—a 5 star open data gold standard knowledge graph of software mentions in scientific articles. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM 2021)*, pages 4574–4583.

David Schindler, Tazin Hossain, Sascha Spors, and Frank Krüger. 2024. A multi-level analysis of data quality for formal software citation.

F. Schroff, D. Kalenichenko, and J. Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823.

L. B. Soares, N. FitzGerald, J. Ling, and T. Kwiatkowski. 2019. Matching the blanks: Distributional similarity for relation learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 2895–2905.

Sharmila Upadhyaya, David Schindler, Frank Krüger, and Stefan Dietze. 2025. SOMD2025: A challenging shared task for software related information extraction. In *Proceedings of the Fifth Workshop on Scholarly Document Processing (SDP 2025)*. Association for Computational Linguistics.

A. van den Oord, Y. Li, and O. Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

M. Vilain, J. Burger, J. Aberdeen, D. Connolly, and L. Hirschman. 1995. A model-theoretic coreference scoring scheme. In *Proceedings of the 6th Conference on Message Understanding (MUC-6)*, pages 45–52.