contains all pseudocode used in this paper. Each algorithm is numbered according to its first appearance in the text.

|  |  |
| --- | --- |
| |  | | --- | | **Algorithm 1:** **Corpus Construction and Cleaning** | |
| **Input:** Raw documents D = {d1, d2, …, dn} |
| **Output:** Cleaned structured corpus C |
| |  | | --- | | 1: For each document di in D: | | 2: a. Apply OCR using ABBYY FineReader 15 → obtain plain text Ti | | 3: b. Apply regex patterns to extract: | | 4: (1) defect manifestation | | 5: (2) root cause | | 6: (3) preventive measure | | 7: c. Remove duplicates and non-informative records | | 8: End For | | 9: Build domain lexicon from Baidu/Sogou dictionaries and HIT stopword list | | 10: Perform tokenization using Jieba with custom dictionary | | 11: Remove stopwords and low-frequency tokens | | 12: Save structured JSON objects: | | 13: { "problem": a, "cause": b, "measure": c } | |

|  |
| --- |
| **Algorithm 2**: **Structured Extraction via Regex and Parsing** |
| **Input:** Raw text content |
| **Output:** DataFrame of quality issues and corresponding causes |
| |  | | --- | | 1: Content = LoadText[FilePath]; | | 2: Content = ReplaceChars[Content, {"【" -> "（", "】" -> "）", " " -> "", "\t" -> ""}]; | | 3: Content = RegexReplace[Content, "[﹄﹃“”]", {"﹄" -> "）", "﹃" -> "（", "“" -> "“", "”" -> "”"}]; | | 4: Titles = RegexFindAll[CompileRegex["\n(\d+\.\d+\.\d+)\s\*([^\n]+?)\s\*(\n|$)\s\*1\s\*[\.。]\s\*现象"], Content]; | | 5: QualityIssues = Map[Join[{#1}, {#2}] &, Titles]; | | 6: CauseBlocks = RegexFindAll[ | | CompileRegex[ | | "原因分析\s\*\n((?:.(?!\s\*(?:原因分析|防治措施|预防措施|\d+[\.。）》]|$))\*.|\n)\*?)" <> | | "(?=\s\*\n\s\*(?:防治措施|预防措施|\d+[\.。）》]|$))" | | ], | | Content | | ]; | | 7: Causes = Map[ | | Module[{Items, Cleaned}, | | Items = RegexFindAll["(?:^|\n)\s\*(?:（?(\d+)[）.)]|(\d+)[\.。、]|)\s\*([^\n]+)", #]; | | Cleaned = Select[Map[TrimText[#3] &, Items], # != "" &]; | | If[Length[Cleaned] > 0, Cleaned, {TrimText[#]}] | | ] &, | | CauseBlocks | | ]; | | 8: Result = MapThread[<|"QualityIssueName" -> #1, "Causes" -> #2|> &, {QualityIssues, Causes}]; | | 9: Do[ | | Print["Quality Issue Name: ", item["QualityIssueName"]]; | | Print["Causes:"]; | | Do[Print[" ", i, ". ", cause], {i, 1, Length[item["Causes"]], cause = item["Causes"][[i]]}]; | | Print["---"], | | {item, Result} | | ]; | | 10: DF = CreateDataFrame[Map[#["Causes"] &, Result]]; | | 11: SaveExcel[DF, OutputPath]; Print["Data successfully saved to: ", OutputPath]; | |

|  |  |
| --- | --- |
| |  | | --- | | **Algorithm 3:** **Semantic Clustering via Hybrid LDA–SBERT Autoencoder** | |
| **Input:** Corpus C, stopwords, domain words, pre-trained SBERT model |
| **Output:** Cluster labels, topic centroids |
| |  | | --- | | 1: FUNCTION PreprocessText(text) | | 2: clean\_text = CleanText(text) | | 3: words = TokenizeAndPOSTag(clean\_text) | | 4: RETURN SelectValidWords(words) | | 5: END FUNCTION | | 6: ProcessedTexts = APPLY PreprocessText TO C | | 7: ValidTexts = FILTER ProcessedTexts WHERE Length > 0 | | 8: Dictionary = BuildDictionary(ValidTexts) | | 9: CorpusData = TextsToCorpus(ValidTexts, Dictionary) | | 10: LDAResults = EMPTY\_LIST | | 11: FOR nTopics = MinTopics TO MaxTopics | | 12: lda\_model = TrainLDA(CorpusData, Dictionary, nTopics, Passes=500, Seed=RandomSeed) | | 13: perplexity = ComputePerplexity(lda\_model, CorpusData) | | 14: coherence = ComputeCoherence(lda\_model, ValidTexts) | | 15: ADD {Topics: nTopics, Perplexity: perplexity, Coherence: coherence, Model: lda\_model} TO LDAResults | | 16: END FOR | | 17: BestLDA = SelectModelWithHighestCoherence(LDAResults) | | 18: Sentences = JoinWords(ValidTexts) | | 19: SBERTEmbeddings = EncodeWithSBERT(PretrainedSBERTModel, Sentences) | | 20: LDAVectors = GenerateLDAVectors(CorpusData, BestLDA.Model) | | 21: CombinedFeatures = Concatenate(SBERTEmbeddings, LDAVectors) | | 22: CombinedScaled = MinMaxScale(CombinedFeatures) | | 23: Autoencoder, Encoder = BuildAutoencoder(InputDim=Dimension(CombinedScaled), LatentDim=AutoencoderDim) | | 24: TrainModel(Autoencoder, CombinedScaled, Epochs=100, BatchSize=64, ValidationSplit=0.2) | | 25: LatentRepresentation = Encoder(Predict(CombinedScaled)) | | 26: ClusterLabels, NumClusters = DensityCanopyKMeans(LatentRepresentation, SampleSize=CanopySampleSize, Seed=RandomSeed) | | 27: VisualizeResults(LatentRepresentation, ClusterLabels, Topics=Range(MinTopics, MaxTopics), | | 28: Perplexities=LDAResults.Perplexity, Coherences=LDAResults.Coherence, | | 29: BestTopic=BestLDA.Topics) | | 30: SaveResults(RawTexts=C, ProcessedTexts=ValidTexts, Labels=ClusterLabels, | | 31: LDAmodel=BestLDA.Model, Corpus=CorpusData) | |

|  |
| --- |
| **Algorithm 4: Causal Network Construction (ISM–BN Coupling)** |
| **Input:** Coded risk factors and defect items |
| **Output:** Directed acyclic causal network (ISM–BN) |
|  |
| 1. Apply Apriori algorithm: |
| Support ≥ 0.05, Confidence ≥ 0.6, Lift > 1 |
| → obtain frequent rules R |
| 2. Construct adjacency matrix A from R |
| 3. Compute reachability matrix: |
| M = (A + I)^k, until M = (A + I)^(k+1) |
| 4. Conduct expert review to refine causal direction and hierarchy |
| 5. Import hierarchy into GeNIe 4.3: |
| - Constrain BN edges to ISM hierarchy |
| - Define binary nodes (Yes/No) |
| 6. Apply Maximum Likelihood Estimation (MLE) + Laplace smoothing |
| to learn CPTs |
| 7. Conduct sensitivity and pathway analysis |
| → identify core risk factors |