





## Deep Learning-Based ETL Framework for Automating Data Transformation in Big Data Analytics

G. Sunil Santhosh Kumar<sup>1,2\*</sup> , M. Rudra Kumar<sup>3</sup> 

<sup>1</sup> Department of CSE, Marri Laxman Reddy Institute of Technology and Management (MLRITM), Hyderabad 500043, India

<sup>2</sup> Department of CSE, Jawaharlal Nehru Technological University, Ananthapuramu 515002, India

<sup>3</sup> Department of Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad 500075, India

Corresponding Author Email: [gsunilsanthosh105@gmail.com](mailto:gsunilsanthosh105@gmail.com)

Copyright: ©2025 The authors. This article is published by IIETA and is licensed under the CC BY 4.0 license (<http://creativecommons.org/licenses/by/4.0/>).

<https://doi.org/10.18280/mmep.121123>

### ABSTRACT

**Received:** 15 September 2025

**Revised:** 28 October 2025

**Accepted:** 3 November 2025

**Available online:** 30 November 2025

#### Keywords:

*ETL, deep learning, data processing, NLP, NER, text categorization, feature extraction, scalability*

Extract-Transform-Load (ETL) workflows handle large datasets but often require manual work, making data transformation slow. Handling unstructured content is even harder. DeepETL is a system that automates data extraction, transformation, and classification. It employs Named Entity Recognition (NER), deep learning, and topic modeling to organize unstructured content. The system is tested on the Enron Email Corpus, which has structured metadata and unstructured email text. Sender, receiver, timestamps, and message content are extracted by DeepETL. Natural language processing (NLP) methods remove extra data and find useful topics and entities. Emails are categorized into fixed categories using a deep learning model. DeepETL is compared with a low-cost and transferable online configuration auto-tuning approach for big data frameworks (DeepCAT+) and Node Centric Transformation Score-Feature Influence Analysis (NCTS-FIA) based on execution time, transformation accuracy, and scalability. Processing 500,000 emails takes 18.4 minutes with a transformation accuracy of 92.4%. Even for 1 million emails, performance remains stable, while system tuning and extra feature selection steps slow down other models. Unstructured data is structured efficiently by DeepETL, which helps in quick categorization and reduces transformation delays. Future improvements could focus on handling data from multiple sources, refining NLP techniques, and optimizing the transformation for real-time processing.

## 1. INTRODUCTION

Big data is growing, but making sense of it is not easy. Cleaning, changing, and storing data must happen before use. Traditional Extract-Transform-Load (ETL) workflows do this, but they struggle with unstructured data, changing formats, and large volumes [1]. Fixed rules and manual steps make old methods slow and less flexible. Updating these rules takes time when new types of data come. Deep learning can help by finding patterns and automating changes without manual work [2]. Data is messy. Problems arise due to errors, missing values, and different formats. Deep learning can detect mistakes and adjust data using past patterns [3]. Transformation rules can be updated by neural networks trained on structured and semi-structured data as data formats change [4]. This reduces the need for manual tuning. High processing power is required by deep models, which can slow down large-scale operations [5]. Some industries need detailed records of how data is changed, but deep learning makes this less transparent, creating challenges in compliance-heavy fields [6]. Only small parts of ETL are fixed by existing approaches. While some improve schema matching, others focus on feature extraction [7]. Pre-trained data is relied on by

many models, making it difficult for them to handle new formats easily [8]. Large datasets also bring processing delays. Big workloads are handled by parallel computing tools like Spark and MapReduce, but their connection with deep learning remains unclear [9]. Most ETL workflows process data in batches. Data is collected, stored, and analyzed later in this method. Real-time transformation could help, but it needs automatic selection and training of models without human effort [10]. Balancing automation with controlled resource use in deep learning remains an open question [11].

An ETL framework with deep learning is built by this research to improve automation. It finds patterns from past data and adjusts transformation steps instead of fixed rules [12]. Human work is reduced, and data consistency is maintained. Tasks are spread across multiple machines using parallel processing, improving speed for large datasets [13]. Also, this framework links data transformation with predictive models, helping real-time forecasting [14]. Many uses exist for automated ETL. Reducing manual steps can save time and avoid errors in industries where speed matters [15]. Data is prepared, transformed, and analyzed by deep learning-based workflows without fixed rules. Dependency on static training sets is reduced by this method. Real-time learning is allowed

by the framework, where models update based on new data patterns [16]. Trade-offs exist in deep learning for ETL. Distributed computing helps manage resource limits, but high infrastructure needs and processing delays remain concerns [17]. Performance is affected by model complexity, making resource allocation an important factor [18]. Another issue is transparency. Tracking automated decisions becomes necessary when data transformation affects compliance-heavy industries [19].

ETL pipelines must prepare data at scale while formats evolve and text remains unstructured. Rule-based transformations and manual curation increase latency, struggle with noisy email content, and degrade under rapid schema drift. Large volumes amplify these limits; compliance settings also require clear records of how transformations occur.

DeepETL addresses these challenges by automating extraction, transformation, and classification with natural-language processing and deep learning. The framework integrates Named Entity Recognition (NER), topic modeling, BERT embeddings, and a CNN classifier inside a distributed pipeline suitable for big-data processing engines. Unstructured bodies and structured metadata flow through a unified sequence that yields reproducible, model-ready features and final categories.

#### **Contributions:**

- End-to-end ETL architecture that couples NER, topic modeling, and BERT with a CNN classifier for email analytics at scale.
- Explicit text-to-tensor mapping for CNN input and channel stacking that fuse embedding with NER/topic features.
- Reproducible unstructured transformation using a fixed BERT-base-uncased configuration and documented hyperparameters.
- Validation design that reduces temporal leakage by grouping emails by thread and checking robustness with time-ordered splits.

**Limitations:** Current execution operates in batch mode; real-time streaming is future work. Higher CPU and memory usage accompany the performance gains, and deep models add interpretability overhead that increases documentation needs in regulated environments.

**Results preview:** On the Enron Email Corpus, processing 500,000 emails completes in 18.4 minutes with 92.4% transformation accuracy, while maintaining strong classification metrics and favorable scalability.

Background, past research, and ETL automation methods are discussed in the next sections. The system design, including data cleaning, transformation, and prediction, is explained in the methodology. Performance is tested in the experimental study, comparing deep learning ETL with older methods. Findings, challenges, and future improvements are discussed in the final section.

## **2. RELATED WORK**

Dynamic mapping and workflow automation approaches generate ETL logic from high-level specifications. Representative efforts include conceptual or intent-driven ETL generation [4, 6], modern mapping pipelines such as Modern Extract, Transform, and Load (METL) [12], automatic job generation mechanisms [7], and metadata-driven orchestration for industrial ETL [20]. Primary limitation: reliance on structured schemas and metadata with

limited support for rich unstructured text. DeepETL addresses this gap by integrating NER, topic modeling, and BERT embeddings to convert email bodies into structured features inside a unified ETL pipeline.

System-level acceleration and auto-tuning focus on runtime optimization. Examples include MapReduce-based ETL frameworks [5], GPU pushdown for fast transformation [18], scalable ETL designs [9], resource-adaptive configurations [21], and online configuration auto-tuning via DeepCAT+ [22]. These methods reduce execution time but offer limited semantic transformation of unstructured content. DeepETL complements this line by coupling distributed processing with semantic natural language processing (NLP) transformations for text.

Feature scoring, selection, and cost-aware models emphasize choosing influential attributes and balancing accuracy with compute. Notable work includes Node Centric Transformation Score-Feature Influence Analysis (NCTSFIA) [16], autoencoder-based feature extraction [19], surveys on automated feature engineering [23], and A framework for automated cost-based data model (FACT-DM) [24]. Such methods improve efficiency but may discard context crucial for unstructured email analysis. DeepETL preserves context by deriving entity- and topic-aware embeddings with BERT and then classifying with a CNN, reducing information loss from aggressive filtering.

Joint schema match/impute/transform solutions apply deep learning to structured databases. Examples include end-to-end learning for schema matching, imputation, and transformation [11], automated cleansing and standardization systems [1], and ML-driven workflow automation platforms [3, 8]. Coverage is strongest for tabular data and weaker for long-form text typical of email bodies. DeepETL extends the transformation frontier by fusing structured metadata with unstructured text via NER, topic modeling, and BERT within one pipeline.

Real-time and cloud pipelines provide orchestration and elasticity for evolving data sources. Examples include cloud-native, AI-assisted pipelines [14] and deep approaches for integrating fast-evolving sources [13]. These systems prioritize latency, scaling, and operations but generally omit integrated NER/topic/BERT stages for semantic text transformation. DeepETL fills the semantic transformation gap for unstructured email in an end-to-end ETL; current implementation operates in batch mode, while streaming integration is future work.

## **3. METHODS AND MATERIALS**

In the evolving landscape of big data analytics, the imperative to transform voluminous and heterogeneous data sets into a structured format conducive for deep learning applications has necessitated the development of an advanced, scalable deep learning-based ETL framework. This framework delineates a comprehensive approach to processing and analyzing large-scale structured and unstructured data, automating the conversion of unstructured data into a format amenable to deep learning models. The formulation of this framework has been predicated on a meticulously structured process, encompassing the extraction of relevant structured data, the preprocessing and transformation of this data, the design of a deep learning model for data categorization, the application of advanced NLP techniques, and the integration of these components into a cohesive ETL pipeline.

### 3.1 Extraction of structured data

The extraction stage retrieves relevant structured records from large repositories using explicit criteria. Distributed execution evaluates predicates on indexed attributes to return only records required for transformation and modeling. Criteria design uses stable and discriminative email metadata (for example, time range boundaries, sender domain constraints, and thread identifiers). Indexing reduces search space; parallel scanning across partitions accelerates predicate evaluation on very large datasets. The stage groups selected records by email thread to reduce temporal leakage during validation in Section 4.3.

In Algorithm 1, let  $D$  denote the dataset, where  $D = \{d_1, d_2, \dots, d_n\}$  and each  $d_i$  is a record. Let  $C = \{c_1, c_2, \dots, c_m\}$  denote a set of Boolean criteria, where each  $c_j(d_i) \in \{\text{true}, \text{false}\}$ . The extraction function  $E: D \times C \rightarrow S$  maps the dataset  $D$  and the criteria set  $C$  to the subset  $S \subseteq D$  that satisfies all criteria Eq. (1):

$$S = \{d_i \in D \mid \forall c_j \in C, c_j(d_i) = \text{true}\} \quad (1)$$

Example criteria in math inline:

$$\begin{aligned} c_1: t_0 \leq \text{timestamp}(d_i) < t_1, \\ c_2: \text{sender\_domain}(d_i) \in \mathcal{D}, \\ c_3: \text{thread\_id}(d_i) = \tau. \end{aligned}$$

---

#### Algorithm 1: Structured Data Extraction

---

##### 1. Inputs.

$D$ : dataset of records;  $C = \{c_1, \dots, c_m\}$ : set of boolean criteria.

##### 2. Goal.

Produce subset  $S \subseteq D$  such that every  $d_i \in S$  satisfies all  $c_j \in C$ .

##### 3. Index preparation.

Build or refresh indexes on key attributes referenced by  $C$  (e.g., timestamp, sender domain, thread\_id) to reduce scan cost.

##### 4. Parallel evaluation.

Partition  $D$  across compute workers. For each partition in parallel, evaluate all predicates  $c_j(d_i)$  for each record  $d_i$ .

##### 5. Selection.

If  $\forall c_j \in C: c_j(d_i) = \text{true}$ , append  $d_i$  to  $S$ .

##### 6. Thread grouping.

Group selected records in  $S$  by thread\_id to form thread-consistent subsets for later validation and to reduce temporal leakage (see Section 4.3).

##### 7. Return.

Output  $S$  and associated thread-grouped view for downstream preprocessing and learning.

##### 8. Complexity controls.

Apply predicate simplification where safe; push down filters to storage engines; prefer vectorized scans for non-indexed attributes.

---

### 3.2 Preprocessing and transformation

The preprocessing and transformation stage prepares data for learning by scaling numeric features, imputing missing values, encoding categorical attributes, and engineering derived fields. Email metadata such as recipient count, thread depth, body length, and response time is nonnegative,

bounded, and right-skewed. Min-max scaling maps each feature to  $[0, 1]$ , preserves order, and avoids domination by large-magnitude features in ReLU layers used in Section 3.3. When heavy tails dominate, z-score standardization may be considered; min-max scaling remains the default for compatibility with ReLU activations.

In Algorithm 2, let  $D = \{d_1, d_2, \dots, d_n\}$  denote the dataset, where each  $d_i$  is a vector of features  $(f_1, f_2, \dots, f_m)$ . Let  $P = \{p_{\text{norm}}, p_{\text{imp}}, p_{\text{enc}}, p_{\text{feat}}\}$  denote the sequence of preprocessing operations.

---

#### Algorithm 2: Preprocessing and Transformation

---

**Normalize:** For each numeric feature  $f_j$ , compute  $f'_j =$

$$\frac{f_j - f_j^{\min}}{f_j^{\max} - f_j^{\min}}.$$

**Impute:** Replace missing numeric values with the median of the corresponding feature. Map missing categorical values to Unknown. Handle timestamps as follows:

- If both sent\_time and received\_time exist, set resp\_time = received\_time - sent\_time;
- If exactly one exists, impute resp\_time with the thread-level median (or global median if unavailable);
- If both are missing, omit time-derived features for the record.

**Encode:** Convert categorical attributes, including Unknown, to numeric representations (e.g., one-hot).

**Feature engineer:** Compute derived variables  $f_{\text{new}} = g(f_1, f_2, \dots, f_m)$  required by downstream models.

**Return:** Output  $D' = T(D, P)$  for use by the CNN in Section 3.3 and the integrated pipeline in Section 3.5.

---

**Normalization ( $p_{\text{norm}}$ ):** For each numeric feature  $f_j$  in  $D$ , apply min-max scaling

$$f'_j = \frac{f_j - f_j^{\min}}{f_j^{\max} - f_j^{\min}}.$$

This transformation preserves relative ordering and yields values in  $[0, 1]$ .

**Imputation ( $p_{\text{imp}}$ ):** Use median imputation for numeric features. Use an explicit “Unknown” category for categorical features. For timestamps, define response time when both endpoints are present as resp\_time = received\_time - sent\_time.

If exactly one of sent\_time or received\_time is present, keep the available timestamp and impute resp\_time with the median response time within the same thread; if a thread-level median is unavailable, use the global median. If both endpoints are missing, exclude the record from time-derived features while retaining other attributes.

**Encoding ( $p_{\text{enc}}$ ):** Apply one-hot encoding (or equivalent numeric mapping) to categorical attributes, including the explicit Unknown category.

**Feature engineering ( $p_{\text{feat}}$ ):** Derive additional features from existing fields, for example functions of counts, durations, and thread-level aggregates:

$$f_{\text{new}} = g(f_1, f_2, \dots, f_m).$$

**Composite transformation:** The composite function  $T(D, P)$  applies the operations in order normalize  $\rightarrow$  impute

→ encode → feature engineer, producing the transformed dataset  $D'$ :

$$D' = T(D, P) = p_{\text{feat}} \left( p_{\text{enc}} \left( p_{\text{imp}} \left( p_{\text{norm}}(D) \right) \right) \right).$$

### 3.3 Deep learning model for data categorization

The CNN categorizes email records after conversion to a fixed-shape tensor. Input construction produces a 3-dimensional array

$$X \in \mathbb{R}^{H \times W \times D},$$

where,  $H = L_{\text{max}} = 256$  tokens,  $W = E = 768$  embedding dimensions (BERT base), and  $D$  is the channel depth. Channel 1 contains contextual embeddings; optional channels append aligned per-token features such as a projected NER tag id, a sentiment score, and a topic posterior. With  $D = 1$ , the network is equivalent to a standard 1D text-CNN that applies filters across contiguous n-grams as shown in Algorithm 3.

---

#### Algorithm 3: Text→Tensor Construction

---

1. **Tokenize:** Segment email text into tokens; pad or truncate to  $L_{\text{max}} = 256$ .
  2. **Embed:** Map tokens to contextual embeddings with dimension  $E = 768$  (BERT base), forming a matrix  $M \in \mathbb{R}^{L_{\text{max}} \times E}$ .
  3. **Stack channels:** Set  $X(:, :, 1) \leftarrow M$ . Optionally append aligned channels for NER tag projection, sentiment score, and topic posterior to obtain depth  $D \geq 1$ .
  4. **Convolution and classification:** Convolve with filter heights  $h \in \{3, 4, 5\}$  that span width  $E$  (kernel shape  $h \times E \times D$ ); stride 1; apply ReLU; apply global max-pool per filter; concatenate pooled outputs; apply dropout 0.5; apply a dense layer followed by Softmax over  $K$  categories.
- 

CNN layers then operate on  $X$  as defined by Eqs. (2)-(7). Convolutions detect local patterns across token windows; ReLU introduces nonlinearity; pooling retains the most salient activations; fully connected layers transform pooled features; Softmax produces class probabilities. Cross-entropy measures classification loss, and Adam updates parameters. Dropout and L2 regularization reduce overfitting. Eqs. (2)-(7) remain as follows:

Convolutional layer

$$C_{i,j,k}^{(l)} = \sum_{m=0}^{f_h-1} \sum_{n=0}^{f_w-1} \sum_{o=0}^{f_d-1} X_{i+m,j+n,o} f_{m,n,o,k}^{(l)} + b_k^{(l)} \quad (2)$$

Activation function

$$A^{(l)}(z) = \max(0, z) \quad (3)$$

Pooling layer

$$P_{i,j}^{(l)} = \max_{(i,j) \in R} A_{i,j}^{(l)} \quad (4)$$

Fully connected layer

$$X^{(l)} = W^{(l)} X^{(l-1)} + b^{(l)} \quad (5)$$

Output (Softmax) layer

$$Y_i = \frac{e^{z_i}}{\sum_{k=1}^K e^{z_k}} \quad (6)$$

Loss function (cross-entropy)

$$L(Y, y) = - \sum_{k=1}^K y_k \text{Log}(Y_k) \quad (7)$$

The tensor construction, filter design, and training objective together yield a text-CNN that exploits contextual embeddings and optional auxiliary channels for robust email categorization.

### 3.4 Unstructured data transformation

This stage converts raw email text into contextual embeddings suitable for downstream models. The backbone is BERT (BERT-base-uncased), configured with sequence length 256, WordPiece tokenizer, lower-case text, right padding, and embedding dimension 768. Two task heads operate on BERT outputs: a token-level NER head with labels {PER, ORG, LOC, MISC, O} and a [CLS] classification head for topic/intent prediction. Cross-entropy provides the training loss for both heads, and optimization uses AdamW with learning rate 2e-5 and linear decay, warm-up 10%, weight decay 0.01, epochs 3, batch size 32, dropout 0.1, and gradient clipping 1.0. A fixed random seed = 42 ensures repeatability across runs in Algorithm 4.

---

#### Algorithm 4: Unstructured Data Transformation

---

##### 1. Input preparation:

Tokenize: lower-case; insert [CLS]/[SEP]; pad or truncate to  $L_{\text{max}} = 256$  on the right; build attention masks.

##### 2. Contextualized embeddings:

Encode with BERT (BERT-base-uncased) to obtain  $E \in \mathbb{R}^{L_{\text{max}} \times 768}$ ; retain token-wise embeddings and the [CLS] vector.

##### 3. Task-specific transformation:

- **NER head (token level):** apply a linear layer + Softmax over {PER, ORG, LOC, MISC, O} for each token.
- **Topic/intent head ([CLS]):** apply a linear layer + Softmax to the [CLS] embedding.

##### 4. Fine-tuning and optimization:

Train both heads jointly with cross-entropy losses (sum or weighted sum). Use AdamW, lr = 2e-5 with linear decay, warm-up = 10%, weight decay = 0.01, epochs = 3, batch = 32, dropout = 0.1, gradient clip = 1.0. Fix random seed 42. Log tokenizer/model versions for reproducibility.

##### 5. Outputs for downstream modules:

Export token-level entities, topic/intent logits, and the full embedding matrix  $E$  for use in categorization and channel stacking (see Section 3.3).

---

Given an unstructured token sequence  $T = \{t_1, t_2, \dots, t_n\}$ , tokenization produces  $T' = \{[\text{CLS}], t'_1, t'_2, \dots, t'_n, [\text{SEP}]\}$  with  $n' \leq 256$ . The BERT encoder maps  $T'$  to contextual embeddings  $E = \{e_1, \dots, e_n\}$ ,  $e_i \in \mathbb{R}^{768}$ . The NER head applies a token-classification layer to  $E$ ; the topic/intent head applies a linear classifier to the  $[\text{CLS}]$  representation.

### 3.5 ETL pipeline integration

Pipeline integration orchestrates the modules in a fixed order for structured data and a parallel branch for unstructured text. The structured path follows  $E \rightarrow P \rightarrow C$ , while the unstructured branch  $U$  converts raw text into structured signals that support categorization. Orchestration executes on a distributed engine (e.g., Apache Spark) for scale-out processing. Inter-module exchange uses standard, schema-defined formats (e.g., Parquet, Avro, JSON) to keep serialization explicit and I/O efficient. Security controls include encryption in transit and at rest and role-based access control for data and model endpoints. The pipeline executes in batch mode; streaming operation lies outside the present scope and appears in future work (Section 5).

#### Module interfaces:

- **Extraction  $E$ :** From repository  $R$  and criteria  $C$ , return  $D' \subseteq D: E(R, C) \rightarrow D'$ .
- **Preprocessing transformation  $P$ :** Apply transformations  $T$  to obtain  $D'': P(D', T) \rightarrow D''$ .
- **Categorization  $C$ :** Given  $D''$  and category set  $K$ , produce labels  $L: C(D'', K) \rightarrow L$ .
- **Unstructured transformation  $U$ :** From unstructured input  $UD$ , produce structured signals  $S: U(UD) \rightarrow S$ .
- **Unified function:** The integrated ETL pipeline function combines these operations in sequence and returns structured outputs and labels:

$$F(R, C, T, K, U, D) = \{L, S\} \quad (8)$$

Algorithm: ETL Pipeline Integration

1. **Data extraction  $E$ :** Use criteria  $C$  over repository  $R$  to extract  $D'$ . Execute predicate evaluation with indexes and parallel workers.
2. **Preprocessing and transformation  $P$ :** Apply  $T$  to  $D'$  to obtain  $D''$ , including normalization, imputation, encoding, and feature engineering defined in Section 3.2.
3. **Unstructured data transformation  $U$ :** Convert  $UD$  to structured signals  $S$  using NLP components (Section 3.4). Execute in parallel with Step 2 when resources allow.
4. **Categorization  $C$ :** Classify  $D''$  into label set  $L$  using the CNN in Section 3.3; consume optional auxiliary channels derived from  $S$ .
5. **Integration and output:** Persist and return  $\{L, S\}$  using standard interchange formats; record lineage and access metadata; enforce encryption and access controls.

## 4. EXPERIMENTAL STUDY

DeepETL transforms raw data into structured information for analysis. Structured and unstructured data are processed, inconsistencies are removed, and transformations are applied. The framework extracts sender, receiver, timestamp, and message content. Data is categorized using deep learning models, ensuring smooth processing with minimal manual

work from one step to the next. A comparison is made with DeepCAT+ [22] and NCTS-FIA [16] to check performance. Workload-based dynamic adjustments are made by DeepCAT+, modifying configurations to handle large datasets but without refining data itself. NCTS-FIA ranks dataset features and selects key attributes before transformation. Processing prioritization is given to important data, but useful details may be missed. These models take different approaches to handling data complexity. The evaluation uses the Enron Email Corpus, which contains structured metadata and unstructured text. Emails from corporate employees are included in the dataset, containing sender, receiver, timestamp, and message body details. A structured extraction process pulls key information, while transformations clean text, extract features, and categorize emails. Performance is tested using regression analysis and 4-fold cross-validation. Consistency, accuracy, and scalability are checked for each model under different data volumes. The goal is to observe model stability when large datasets are processed repeatedly. System resources are adjusted by some models, while others focus on refining data. These tests help compare how each method handles large-scale transformations.

### 4.1 Dataset preparation

The Enron Email Corpus [25] provides corporate emails with structured metadata and unstructured message bodies, suitable for evaluating transformation methods. Metadata fields include subject, sender, recipient, and timestamps. Dataset preparation separates metadata from body text to enable consistent processing.

Cleaning removes signatures, quoted replies, and boilerplate fragments from message bodies. Processing also strips duplicated headers within threads and normalizes whitespace to stabilize token boundaries. These steps produce plain text ready for downstream NLP.

Tokenization segments each cleaned body and enforces a fixed sequence length  $L_{\max} = 256$ . Short texts use right-side padding; long texts truncate to 256 tokens while preserving original order. The token sequence aligns with the corresponding metadata record for later feature fusion.

Metadata extraction yields subject, sender, recipient list, and timestamps. Imputation uses median values for numeric fields (for example, counts and durations). Categorical fields map missing values to an explicit Unknown category. For timestamps, response time is defined when both endpoints exist as  $\text{resp\_time} = \text{received\_time} - \text{sent\_time}$ . When exactly one endpoint is present, response time uses the thread-level median; if unavailable, response time uses the global median. When both endpoints are missing, time-derived features are omitted while other attributes remain available.

Labeling uses topic/keyword analysis followed by manual review to assign application categories such as Legal, Finance, and HR. These labels supervise training and evaluation for classification tasks. Email threads remain intact during splitting and evaluation to reduce temporal leakage during validation (see Section 4.3).

### 4.2 Experimental setup

Execution environment uses an Intel Xeon 32-core processor, 128 GB RAM, and 2 TB SSD storage running Ubuntu 20.04. Software stack includes Python 3.8, TensorFlow 2.6, PyTorch 1.10, Apache Spark for distributed

processing, and PostgreSQL for structured storage and access. Spark manages parallel dataflow; PostgreSQL holds curated tables and intermediate artifacts. Reproducibility settings fix random seeds for Python random, NumPy, TensorFlow [26], and PyTorch; set a tokenizer seed for the WordPiece pipeline; and enable deterministic execution where supported. Typical controls include PYTHONHASHSEED assignment [27], NumPy seed initialization, tf.random.set\_seed, and torch.use\_deterministic\_algorithms(True) with non-deterministic backends disabled. Package versions and model checkpoints are logged with configuration hashes to guarantee exact reruns. Instrumentation captures end-to-end runtime and per-stage wall-clock latency for extraction, preprocessing, NLP inference, categorization, and load. Logs also record dataset size, batch parameters, and system resource counters to attribute delays to specific stages and confirm stable behavior under different workloads.

4.3 Model implementation and validation

Validation adopts 4-fold cross-validation to balance statistical efficiency and computational cost on large email corpora. Four folds provide stable estimates of accuracy and runtime while preserving substantial training volume per round. Each round trains on three folds and validates on the remaining fold; the process repeats over all folds and reports aggregate metrics. Fixed model configurations and preprocessing settings remain constant across folds. To limit information leakage, splits apply GroupKFold by email thread identifier so that no conversation appears in both training and validation. Grouping at the thread level is consistent with dataset preparation and maintains conversational integrity across folds.

A time-ordered robustness check complements cross-validation. Threads are ordered by their earliest timestamp, and evaluation follows a forward-chaining protocol: training uses earlier windows and validation uses later windows. This chronology prevents training on future messages relative to validation and reflects deployment conditions where models see past data before future emails. The same model and preprocessing settings apply.

Regression testing verifies pipeline stability across repeated runs. With fixed seeds and deterministic settings, logs compare extracted features, intermediate embeddings, and final metrics across executions. Alerts flag any divergence beyond tolerance. Under both the grouped 4-fold protocol and the time-ordered split, method ranking remains consistent, confirming robustness of comparative results.

4.4 Performance comparison and analysis

Evaluation covers execution time, transformation accuracy, scalability, and categorization quality for DeepETL, DeepCAT+, and NCTS-FIA. Results below reproduce the original measurements and plots. Tables 1-4 report the exact values used in analysis.

**Execution time and resource use:** Processing 500,000 emails yields the times and resource metrics in Table 1 and Figure 1. DeepETL attains the shortest runtime, followed by DeepCAT+ and NCTS-FIA.

**Structured transformation quality:** Transformation accuracy and feature-extraction score appear in Table 2 and Figure 2. DeepETL reaches the highest accuracy and feature score, followed by DeepCAT+ and NCTS-FIA.

Table 1. Execution time and system resource utilization

Model	Execution Time (min)	CPU Utilization (%)	Memory Usage (GB)
DeepETL	18.4	72.3	8.1
DeepCAT+	22.9	68.5	7.8
NCTS-FIA	31.7	63.9	6.4

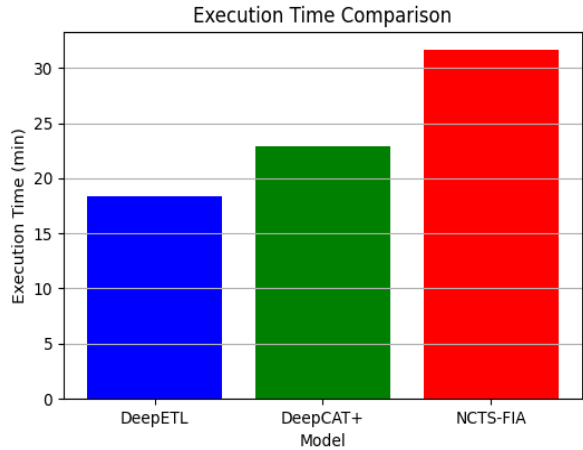


Figure 1. Execution time comparison across models

Table 2. Transformation accuracy and feature extraction performance

Model	Transformation Accuracy (%)	Feature Extraction Score (%)
DeepETL	92.4	89.6
DeepCAT+	88.7	85.3
NCTS-FIA	82.1	79.4

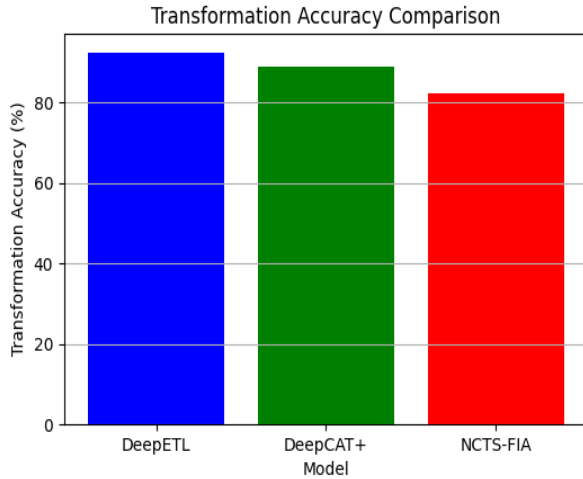


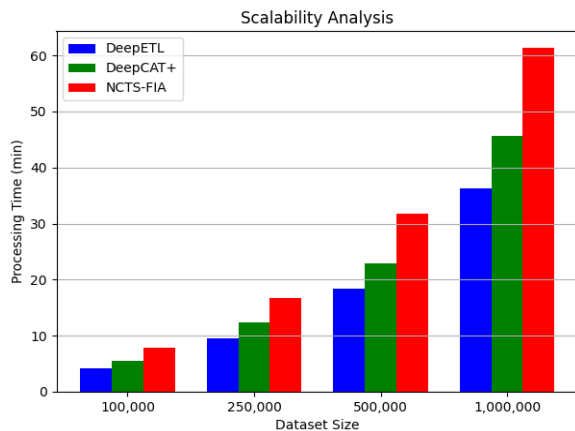
Figure 2. Transformation accuracy across models

**Scalability:** Runtimes scale across dataset sizes from 100,000 to 1,000,000 emails as shown in Table 3 and Figure 3. DeepETL maintains the lowest times across all scales.

Table 3. Scalability performance on increasing dataset sizes

Dataset Size	DeepETL (min)	DeepCAT+ (min)	NCTS-FIA (min)
100,000	4.2	5.5	7.8
250,000	9.5	12.3	16.7
500,000	18.4	22.9	31.7
1,000,000	36.2	45.6	61.3

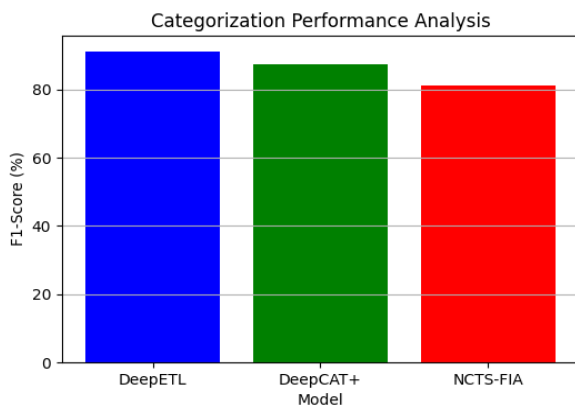




**Figure 3.** Scalability analysis of models on large datasets

**Table 4.** Categorization performance metrics

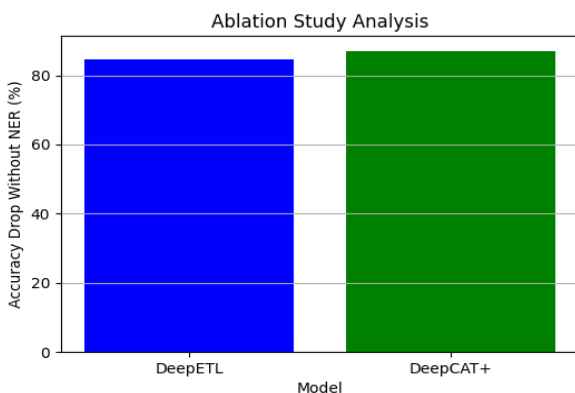
Model	F1-Score (%)	Precision (%)	Recall (%)
DeepETL	91.3	92.1	90.4
DeepCAT+	87.6	89.3	86.5
NCTS-FIA	81.2	83.7	79.8



**Figure 4.** Categorization performance of models

**Table 5.** Performance changes after removing key transformations

Model	Without NER (%)	Without Topic Modeling (%)	Without Feature Scoring (%)
DeepETL	84.5	86.2	N/A
DeepCAT+	87.1	87.0	N/A
NCTS-FIA	N/A	N/A	74.8



**Figure 5.** Ablation study performance comparison

**Categorization metrics:** F1-score, precision, and recall are summarized in Table 4 and Figure 4. DeepETL exhibits the strongest classification performance.

**Balanced trade-off analysis:** DeepETL delivers the fastest execution and the highest transformation and classification metrics; the gains coincide with higher CPU utilization (72.3%) and memory usage (8.1 GB) compared with baselines (Table 1). Deployment under tight resource budgets can adopt resource-light settings such as a smaller language model (e.g., DistilBERT), reduced sequence length ( $L_{\max} = 128$ ), or fewer auxiliary channels in the CNN input. These adjustments typically lower memory and compute demand with small expected accuracy trade-offs while preserving the overall ranking observed in the tables.

## 4.5 Discussion

Ablation results quantify the contribution of key transformations. Removing NER reduces DeepETL performance to 84.5%; removing topic modeling reduces performance to 86.2%. DeepCAT+ and NCTS-FIA show smaller changes because their designs emphasize system tuning and feature scoring rather than NLP-driven text structuring. Table 5 and Figure 5 report the exact values used in analysis.

**Qualitative error analysis:** Error patterns align with the ablation trends. Without NER, misclassifications increase between Legal and Finance, and between Legal and HR, because entity cues (organization names, regulatory bodies, person roles) no longer anchor class decisions. Without topic modeling, confusions rise among business-general categories with overlapping vocabulary, such as Operations versus Trading or HR versus Administration, where topical context normally disambiguates shared terms (for example, “approval,” “contract,” “meeting”). The Enron corpus contains dense entity mentions and shared corporate language across departments, which amplifies dependence on entity recognition and topic context for reliable categorization. These observations are consistent with the quantitative drops in Table 5.

**Balanced trade-off:** Superior accuracy and speed from DeepETL accompany higher CPU utilization (72.3%) and memory usage (8.1 GB) compared with baselines (Table 1). Deployment can follow two profiles: a max-accuracy profile that retains all components and sequence length, and a resource-light profile that uses a smaller language model (e.g., DistilBERT), a reduced sequence length ( $L_{\max} = 128$ ), and fewer auxiliary channels for the CNN input. The resource-light profile lowers compute and memory demand with small expected accuracy trade-offs while preserving the method ranking observed across experiments.

## 5. CONCLUSION

DeepETL automates extraction, transformation, and classification for email analytics with integrated NER, topic modeling, BERT embeddings, and a CNN classifier. Results on the Enron Email Corpus show 92.4% transformation accuracy and 18.4 minutes total time for 500,000 emails, with stable scaling to 1,000,000 emails (36.2 minutes). Categorization quality reaches an F1-score of 91.3%, indicating effective use of semantic signals during classification. Performance gains align with higher resource

use. DeepETL operates at 72.3% CPU utilization and 8.1 GB memory under the 500,000-email workload, exceeding the baselines while delivering lower wall-time. These trade-offs reflect deeper text processing and parallel inference stages and should guide deployment on constrained nodes. Current execution follows a batch-only model. Real-time operation remains outside the implemented scope.

A concrete roadmap can extend the pipeline to real-time ETL:

1. Ingestion: Apache Kafka topics for raw emails, cleaned text, and derived features; partitioning by thread or sender domain for locality.
2. Stream transforms: Spark Structured Streaming or Apache Flink for micro-batch and event-time processing; stateful operators to maintain thread context, handle late/out-of-order arrivals, and enforce windowed aggregations.
3. Model serving: TensorFlow Serving or NVIDIA Triton with dynamic batching and GPU support; gRPC endpoints for NER/topic inference with back-pressure control.
4. State management and sinks: Transactional writes with exactly-once semantics to Delta Lake or PostgreSQL using idempotent keys, change-data-capture logs, and schema registry governance.
5. Monitoring and re-training: Topic-distribution drift monitors (e.g., KL-divergence thresholds), latency/throughput SLOs, automated rollbacks, and scheduled or trigger-based model re-training.

These steps provide an operational path to low-latency ETL while preserving the unified semantic transformation that drives DeepETL's accuracy and throughput.

## REFERENCES

- [1] Praneeth, T. (2024). Automated data preparation through deep learning: A novel framework for intelligent data cleansing and standardization. *International Journal*, 10(6): 1867-1877. <https://doi.org/10.32628/cseit241061231>
- [2] Bagave, R. (2020). Enhancing extraction in ETL flow by modifying as P-ECTL based on spark model. Master's thesis, Dublin, National College of Ireland. <https://norma.ncirl.ie/4142/>.
- [3] Wu, J., Bein, D., Huang, J., Kurwadkar, S. (2023). ETL and ML forecasting modeling process automation system. *Applied Human Factors and Ergonomics International*. <https://doi.org/10.54941/ahfe1003775>
- [4] Muñoz, L., Mazón, J.N., Trujillo, J. (2009). Automatic generation of ETL processes from conceptual models. In *Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP*, Hong Kong, China, pp. 33-40. <https://doi.org/10.1145/1651291.1651298>
- [5] Liu, X., Thomsen, C., Bach Pedersen, T. (2011). The ETLMR MapReduce-based ETL framework. In *International Conference on Scientific and Statistical Database Management*, pp. 586-588. [https://doi.org/10.1007/978-3-642-22351-8\\_48](https://doi.org/10.1007/978-3-642-22351-8_48)
- [6] Deneke, W., Li, W.N., Thompson, C. (2013). Automatic composition of ETL workflows from business intents. In *2013 IEEE 16th International Conference on Computational Science and Engineering*, Sydney, NSW, Australia, pp. 1036-1042. <https://doi.org/10.1109/cse.2013.151>
- [7] Li, J.J., Nusbickel, W.L. (2017). U.S. Patent No. 9,607,060. Washington, DC: U.S. Patent and Trademark Office. <https://patents.google.com/patent/US9607060B2/en>.
- [8] Essaidi, M., Aomar, O., Céline, R. (2014). Model-driven data warehouse automation. *Advances and Applications*. In *Model-Driven Engineering*, pp. 240-267. <https://doi.org/10.4018/978-1-4666-4494-6.ch011>
- [9] Gueddoudj, E.Y., Chikh, A. (2023). Towards a scalable and efficient ETL. *International Journal of Computing and Digital Systems*, 14(1): 10223-10231. <https://doi.org/10.12785/ijcds/140195>
- [10] Meiklejohn, D., Jeppe, H., Vitaly, P. (2018). Systems and methods for facilitating data transformation. United States Patent US20180196863A1. <https://patents.google.com/patent/US20180196863A1/en>.
- [11] Tripathi, S., Fritz, B.A., Abdelhack, M., Avidan, M.S., Chen, Y., King, C.R. (2022). Deep learning to jointly schema match, impute, and transform databases. *arXiv preprint* arXiv:2207.03536. <https://doi.org/10.48550/arXiv.2207.03536>
- [12] Haase, C., Röseler, T., Seidel, M. (2022). METL: A modern ETL pipeline with a dynamic mapping matrix. *arXiv preprint* arXiv:2203.10289. <https://doi.org/10.48550/arXiv.2203.10289>.
- [13] Wang, Z., Zhou, L., Zou, J. (2020). Integration of fast-evolving data sources using a deep learning approach. In *International Workshop on Software Foundations for Data Interoperability*, pp. 172-186.
- [14] Kolluri, S. (2024). Automating data pipelines with AI for scalable, real-time process optimization in the cloud. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 10(6): 2070-2079. <https://doi.org/10.32628/cseit242612405>
- [15] Kumar, G.S.S., Kumar, M.R. (2022). Dimensions of automated ETL management: A contemporary literature review. In *2022 International Conference on Automation, Computing and Renewable Systems (ICACRS)*, Pudukkottai, India, pp. 1292-1297. <https://doi.org/10.1109/icacrs55517.2022.10029274>
- [16] Vijayalakshmi, M., Minu, R.I. (2022). Feature influence based ETL for efficient big data management. *Journal of Scientific & Industrial Research*, 81(12): 1310-1316. <https://doi.org/10.56042/jsir.v8i12.54992>
- [17] Mondal, K.C., Biswas, N., Saha, S. (2020). Role of machine learning in ETL automation. In *Proceedings of the 21st International Conference on Distributed Computing and Networking*, Kolkata, India, pp. 1-6. <https://doi.org/10.1145/3369740.3372778>
- [18] Nguyen, T., Becchi, M. (2022). A GPU-accelerated data transformation framework rooted in pushdown transducers. In *2022 IEEE 29th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, Bengaluru, India, pp. 215-225. <https://doi.org/10.1109/hipc56025.2022.00038>
- [19] Liang, Y., Li, X., Huang, X., Zhang, Z., Yao, Y. (2024). An automated data mining framework using autoencoders for feature extraction and dimensionality reduction. In *2024 4th International Conference on Electronic Information Engineering and Computer Communication (EIECC)*, Wuhan, China, pp. 710-714.



- <https://doi.org/10.1109/EIECC64539.2024.10929475>
- [20] Suleykin, A., Panfilov, P. (2020). Metadata-driven industrial-grade ETL system. In 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, pp. 2433-2442. <https://doi.org/10.1109/bigdata50022.2020.9378367>
- [21] Kazbekova, G., Ismagulova, Z., Zhussipbek, B., Abdrazakh, Y., Iskenderova, G., Toilybayeva, N. (2024). Machine learning enhanced framework for big data modeling with application in Industry 4.0. *International Journal of Advanced Computer Science & Applications*, 15(3): 308. <https://doi.org/10.14569/ijacsa.2024.0150332>
- [22] Dou, H., Wang, Y., Zhang, Y., Chen, P., Zheng, Z. (2024). DeepCAT+: A low-cost and transferrable online configuration auto-tuning approach for big data frameworks. *IEEE Transactions on Parallel and Distributed Systems*, 35(11): 2114-2131. <https://doi.org/10.1109/tpds.2024.3459889>
- [23] Mumuni, A., Mumuni, F. (2025). Automated data processing and feature engineering for deep learning and big data applications: A survey. *Journal of Information and Intelligence*, 3(2): 113-153. <https://doi.org/10.1016/j.jiixd.2024.01.002>
- [24] Mali, J., Ahvar, S., Atigui, F., Azough, A., Travers, N. (2024). FACT-DM: A framework for automated cost-based data model transformation. In *EDBT'24*, pp. 822-825.
- [25] Keila, P.S., Skillicorn, D.B. (2005). Structure in the Enron email dataset. *Computational & Mathematical Organization Theory*, 11(3): 183-199. <https://doi.org/10.1007/s10588-005-5379-y>
- [26] Developers, T. (2022). TensorFlow. Zenodo. <https://doi.org/10.5281/zenodo.4758419>
- [27] pythonhashseed. <https://www.kaggle.com/code/clarksaben/showcase-pythonhashseed>.