Supplementary Materials: Data Processing Pipeline and Experimental Reproducibility

1 Overview

This document provides complete technical documentation for reproducing the experimental results reported in the main manuscript. All data processing scripts, trained models, and evaluation outputs are organized in structured directories to ensure full transparency and reproducibility.

2 S1. Data Post-Processing Pipeline

2.1 S1.1 Pipeline Architecture

The ETCBC encoding system produces morphologically rich output strings that require systematic transformation before evaluation. We implement a four-stage processing pipeline to convert raw model predictions into standardized evaluation formats:

Stage	Transformation
Stage 1	Model Output → Reduced Format
Stage 2	Pattern Extraction → Mapping Table Generation
Stage 3	Reduced Format → Final Format (ID-annotated)
Stage 4	Final Format → Pure Label Sequence

Stage 1: Symbol Pair Reduction **Script:** `01_out_original_reducer.py`

The ETCBC system uses redundant delimiter pairs (`!!`, `@@`, `]]`) to enclose morphological markers. This stage removes the opening delimiter while preserving content and closing delimiters:

- `!content!` → `content!`
- `@content@` → `content@`
- `]content]` → `content]`

This unification eliminates syntactic redundancy without losing morphological information.

Stage 2: Pattern Vocabulary Construction **Script:** `02_generate_patterns_csv_from_reduced.py`

This stage extracts all unique morphological patterns from the reduced output and generates a mapping table (`patterns.csv`) that assigns each pattern a unique integer ID (0–328 for the S2+S3+S4 corpus). The pattern vocabulary exhibits a steep power-law distribution:

- Empty pattern (null morphology): 50.5% of tokens

- Word boundary marker (-): 6.6%
- Nominal ending (/): 2.7%
- Top 100 patterns: >99.9% coverage
- Rare complex patterns (e.g., =/~, !(>(T&): <10 occurrences each

Stage 3: Label Encoding **Script:** 03_convert_reduced_to_final.py

This stage converts the reduced format into an ID-annotated format where each character is preceded by its morphological label ID:

```
Input (reduced):  WBHL JN
Output (final):    0W1B1H0 0J0N0
```

Here, 0 represents the null pattern, 1 represents a specific morphological marker, etc. This format preserves character-level alignment while encoding morphological information.

Stage 4: Label Extraction **Script:** 04_convert_final_to_out.py

The final stage extracts pure label sequences for metric calculation:

```
Input (final):  0W1B1H0 0J0N0
Output (.out):  0 1 1 0 0 0 0
```

Critical invariant: The number of labels equals the number of characters plus the number of words (word-final markers). This ensures complete morphological annotation.

2.2 S1.2 File Format Specifications

Input Format (.in files) Raw Syriac text in ASCII transliteration, space-delimited:

```
WQM MNWX W>ZL <M >NTTH W>T> LWT GBR>
```

Each line contains a variable number of words (typically 7 in our corpus structure).

Output Format (.out files) Space-delimited integer label sequences:

```
0 1 31 3 0 0 0 0 2 0 1 0 0 3 0 0 0 0 0 0 13 1 0 0 1 0 10 3 0 0 0 0
```

Labels range from 0 to 328 (329 total classes including the null pattern).

Pattern Mapping Table (patterns.csv) CSV file mapping pattern strings to label IDs:

```
label,pattern,count
0,,3066705
1,-,404156
2,/,165508
...
290,<UNKNOWN>,0
```

The count field records empirical frequency in the training corpus. Patterns with count=0 are reserved for unseen morphological combinations.

3 S2. Experimental Results Organization

All experimental results are stored in the `report/` directory, organized by model architecture and hyperparameter configuration. The directory structure follows a strict naming convention to ensure self-documentation:

3.1 S2.1 Directory Structure

```
report/
|-- encoder_decoder_30ep_bs128_lr1e-4_emb512_h8_d0.1_b3_s7/
|   |-- s2_on_s2/
|   |   |-- MODEL_s2-in_s2-out_ONE_DATASET/
|   |   |   |-- seq2seq_*.pth
|   |   |   |-- model_config*.json
|   |   |-- results/
|   |   |   |-- results_*.txt
|   |   |   |-- test.out.original.truth
|   |   |   |-- test.out.reduced
|   |   |   |-- test.out.final
|   |   |   |-- test.out
|   |   |-- data_s2_on_s2/
|   |       |-- train.in, train.out
|   |       |-- val.in, val.out
|   |       |-- test.in, test.out
|   |       |-- patterns.csv
|   |-- s4_on_s2/
|       |-- (same structure as s2_on_s2)
|
|-- encoder_only_100ep_bs128_lr3e-4_d512_l4_h16_d0.25_5seeds/
|   |-- s2_on_s2/
|   |   |-- transformer_train_*_seed123/
|   |   |   |-- data/
|   |   |   |-- results/
|   |   |   |   |-- transformer_predictions_*.out
|   |   |   |   |-- transformer_results_*.json
|   |   |   |   |-- restore_and_levenshtein/
|   |   |   |       |-- levenshtein_results.json
|   |   |   |-- plots/
|   |   |   |-- tensorboard/
|   |   |   |-- training_history.json
|   |   |   |-- transformer_model.pt
|   |   |-- levenshtein_summary.json
|   |   |-- levenshtein_summary.txt
|   |-- s4_on_s2/
|       |-- (same structure)
|
|-- mdlm_200ep_bs16_lr1e-4_d768_l5_h4_d0.25_steps2_5seeds/
|   |-- s2_on_s2/
|   |-- s4_on_s2/
|
|-- case_study_mdlm_50ep_bs16_lr3e-4_d384_l6_h4_d0.25_steps2_s52/
    |-- s2_on_s2/
```

3.2 S2.2 Naming Convention

Directory names encode complete hyperparameter configurations:

Component	Meaning
`encoder_only`	Model architecture (encoder-only Transformer)
`100ep`	Maximum training epochs
`bs128`	Batch size
`lr3e-4`	Learning rate (3×10^{-4})
`d512`	Model hidden dimension
`l4`	Number of layers
`h16`	Number of attention heads
`d0.25`	Dropout rate
`5seeds`	Number of random seeds used

This self-documenting convention eliminates ambiguity about experimental conditions.

3.3 S2.3 Data Configuration: s2_on_s2 vs s4_on_s2

Each model is evaluated under two data regimes:

s2_on_s2 (Baseline)

- **Training set:** S2 corpus (50,649 lines)
- **Test set:** S2 test split (10,869 lines)
- **Purpose:** Establish in-distribution performance ceiling

s4_on_s2 (Target-Specific Scaling)

- **Training set:** S4 corpus (97,048 lines, 92% larger)
- **Test set:** S2 test split (10,869 lines, *identical to s2_on_s2*)
- **Purpose:** Evaluate whether heterogeneous data expansion aids target domain performance

Critical design choice: Both configurations use the *exact same* validation and test sets, enabling controlled comparison of training data effects. This setup corresponds to Method 3 (Target-Specific Split) in the main manuscript (Section 3.2).

3.4 S2.4 Multi-Seed Experimental Protocol

To ensure statistical robustness, experiments use 5 independent random seeds:

```
{123, 456, 789, 2023, 2024}
```

Each seed controls:

- Model parameter initialization
- Training data shuffling order

- Dropout mask sampling
- Validation set partitioning (when applicable)

Results are reported as **mean ± standard deviation** across seeds. Seeds showing outlier performance (>2σ from median) are excluded and documented in `levenshtein_summary.json`.

3.5 S2.5 Key Result Files

Per-Seed Results Each seed produces:

- `transformer_model.pt`: Trained model checkpoint
- `training_history.json`: Epoch-by-epoch loss/accuracy
- `transformer_results_*.json`: Classification report (precision/recall/F1 per class)
- `levenshtein_results.json`: Character Error Rate (CER) metrics

Aggregated Results `levenshtein_summary.json` contains cross-seed statistics:


```
{
  "dataset": "s2_on_s2",
  "num_seeds": 5,
  "excluded_seeds": ["1024", "42"],
  "individual_results": [
    {
      "seed": "123",
      "char_accuracy": 0.9611,
      "line_accuracy": 0.4103,
      "avg_distance": 2.1786,
      "exact_matches": 4460,
      "total_lines": 10869
    }
  ],
  "statistics": {
    "char_accuracy_mean": 0.9612,
    "char_accuracy_std": 0.0003,
    "line_accuracy_mean": 0.4096,
    "line_accuracy_std": 0.0012
  }
}
```


4 S3. Evaluation Metrics

4.1 S3.1 Primary Metric: Character Error Rate (CER)

The primary evaluation metric is the Levenshtein-based Character Error Rate:

$$\text{CER} = \frac{\text{EditDistance}(\text{predicted}, \text{gold})}{\text{Length}(\text{gold})} \times 100\%$$

where $\text{EditDistance}(\cdot, \cdot)$ is the minimum number of character-level insertions, deletions, and substitutions required to transform the predicted sequence into the gold standard.

Rationale for CER Unlike word-level accuracy (which penalizes partial errors equally to complete failures), CER provides a granular measure of annotation quality that directly reflects post-editing effort in real-world scholarly workflows. A model with CER = 3.42% requires, on average, 3.42 character corrections per 100 charactersa concrete metric for human labor savings.

4.2 S3.2 Secondary Metrics

Line-Level Exact Match Accuracy

$$\text{LineAcc} = \frac{\text{Num. lines with CER} = 0}{\text{Total lines}} \times 100\%$$

This strict metric quantifies the proportion of sequences requiring no post-editing.

Average Edit Distance

$$\text{AvgDist} = \frac{1}{N} \sum_{i=1}^{N} \text{EditDistance}(\text{pred}_i, \text{gold}_i)$$

This unnormalized metric captures absolute error magnitude per sequence.

4.3 S3.3 Multi-Dimensional Accuracy Analysis

The supplementary evaluation toolkit (`data_post_processing/accuracy_analysis_system/`) implements a 2×2 matrix analysis that stratifies performance by input/output novelty:

	Output ∈ Train	**Output ∉ Train**
Input ∈ Train	Memorization	Generalization-1
Input ∉ Train	Generalization-2	True Generalization

This decomposition reveals whether models rely on rote memorization versus compositional understanding. Results are documented in `accuracy_2d_results.txt`.

5 S4. Reproducibility Protocol

5.1 S4.1 Software Environment

- **Operating System:** Ubuntu 22.04 LTS
- **Python Version:** 3.10.12
- **PyTorch Version:** 2.0.1
- **CUDA Version:** 11.8
- **GPU:** NVIDIA RTX 4090D (24GB VRAM)

All dependencies are specified in `requirements.txt`.

5.2 S4.2 Data Preparation

Step 1: Obtain ETCBC Corpus Download the publicly available Syriac morphological database:

```
git clone https://github.com/ETCBC/ssi_morphology
```

Step 2: Dataset Configuration Two pre-processed configurations are provided:

- `data/s2_on_s2/`: Standard configuration (50,649 training lines)
- `data/s4_on_s2/`: Extended configuration (97,048 training lines)

Each directory contains:

```
train.in, train.out
val.in, val.out
test.in, test.out
patterns.csv
```

5.3 S4.3 Training Reproduction

To exactly reproduce the Encoder-Only results on s4_on_s2:

```
for seed in 123 456 789 2023 2024; do
    python train.py \
        --model_type transformer \
        --data_dir data/s4_on_s2 \
        --seed $seed \
        --batch_size 128 \
        --learning_rate 3e-4 \
        --hidden_dim 512 \
        --num_layers 4 \
        --num_heads 16 \
        --dropout 0.25 \
        --max_epochs 100 \
        --output_dir report/encoder_only_reproduction
done
```

Training time per seed: approximately 4.8 hours on RTX 4090D.

5.4 S4.4 Evaluation Reproduction

After training, run the evaluation pipeline:

```
# Stage 1: Symbol reduction
python data_post_processing/decompose/01_out_original_reducer.py \
    --input results/predictions.out.original \
    --output results/predictions.out.reduced

# Stage 2: Generate pattern mapping (if needed)
python data_post_processing/decompose/02
   _generate_patterns_csv_from_reduced.py \
    --input results/predictions.out.reduced \
    --output patterns_new.csv
```

```
# Stage 3: Convert to final format
python data_post_processing/decompose/03_convert_reduced_to_final.py \
    --input results/predictions.out.reduced \
    --patterns data/s4_on_s2/patterns.csv \
    --output results/predictions.out.final

# Stage 4: Extract labels
python data_post_processing/decompose/04_convert_final_to_out.py \
    --input results/predictions.out.final \
    --output results/predictions.out

# Compute CER
python data_post_processing/accuracy_analysis_system/accuracy_analyzer.py
   \
    --gold data/s4_on_s2/test.out \
    --pred results/predictions.out
```

5.5 S4.5 Expected Outputs

Successful reproduction should yield CER values within 0.1% of reported means:

Model	s2_on_s2 CER	s4_on_s2 CER
Transformer (Baseline)	4.00% ± 0.05	4.08% ± 0.06
Encoder-Only	3.88% ± 0.03	3.53% ± 0.04
MDLM	3.64% ± 0.05	3.42% ± 0.08

Deviations beyond ±0.15% may indicate environmental differences (e.g., PyTorch version, hardware floating-point precision).

6 S5. Case Study: Two-Step MDLM Analysis

The case study reported in Section 3.5 of the main manuscript uses a specialized configuration:

- **Directory:** `case_study_mdlm_50ep_bs16_lr3e-4_d384_l6_h4_d0.25_steps2_s52`
- **Configuration:** 2-step diffusion ($T = 2$), deliberately undertrained (50 epochs)
- **Purpose:** Enable meaningful error analysis by preserving sufficient failure cases
- **Seed:** Fixed at 52 for deterministic analysis

This configuration achieves 85.38% overall accuracy (vs. 96%+ for production models), providing a balance between competence and interpretability. Detailed per-step accuracy breakdowns are stored in `results/step_analysis.json`.

7 S6. Data Availability Statement

All experimental artifacts are publicly accessible:

- **ETCBC Syriac Corpus:** `https://github.com/ETCBC/ssi_morphology`
- **Model Code & Trained Checkpoints:** `https://anonymous.4open.science/r/mdlm`
- **Processing Scripts:** Included in `data_post_processing/` directory
- **Full Results Archive:** `report/` directory (approximately 15GB)

Upon publication, the anonymized repository will be migrated to a permanent institutional archive with DOI assignment.

8 S7. Computational Cost

Total computational budget for all experiments:

- Baseline Transformer: 6.2h $\times$ 1 seed $\times$ 2 regimes = 12.4 GPU-hours
- Encoder-Only: 4.8h $\times$ 5 seeds $\times$ 2 regimes = 48 GPU-hours
- MDLM: 7.4h $\times$ 5 seeds $\times$ 2 regimes = 74 GPU-hours
- Case Study MDLM: 3.5h $\times$ 1 seed = 3.5 GPU-hours
- **Total:** Approximately 138 GPU-hours

At current cloud GPU pricing ($1.50/hour for RTX 4090 equivalent), total compute cost $\approx$ $207 USD.