

Lossy Orthography and Linguistic Inference: Computational Approaches to Arabic Rasm Restoration

Maged S. Al-Shaibani¹ Irfan Ahmad^{1,2}

¹SDAIA-KFUPM Joint Research Center for AI, KFUPM

²Information and Computer Science Department, KFUPM

Abstract

Restoring dots to dotless Arabic text, historically known as *rasm*, is a foundational yet underexplored problem in Arabic NLP. While Arabic diacritization focuses on recovering omitted vowels, rasm restoration addresses a deeper level of orthographic ambiguity in which consonantal distinctions themselves are absent from the written text. Beyond its linguistic and historical significance, the task has applications in representation learning, tokenization, OCR, manuscript digitization, information retrieval, and content moderation. In this paper, we present a large-scale comparative study of Arabic rasm restoration across statistical, neural, pretrained, and prompt-based paradigms. We evaluate n-gram language models, BiLSTMs and Transformers trained from scratch, pretrained character-level CANINE models, and large language models (LLMs) using few-shot prompting. Our experiments were conducted on a diverse corpus exceeding 431M words spanning social media, news, religious texts, poetry, and academic writing. Results show that Transformer models achieve the best overall performance while remaining computationally efficient. BiLSTMs perform competitively, and classical n-gram models remain surprisingly strong despite their simplicity. In contrast, modern LLMs perform substantially below specialized models, with flagship systems only matching the performance of a 4-gram baseline. These findings position rasm restoration as a challenging benchmark for contextual orthographic inference and a natural extension of Arabic diacritization research.

1 Introduction

Arabic writing provides a particularly illuminating example of how human language can tolerate substantial orthographic underspecification while remaining efficiently decodable by expert readers. In modern Arabic orthography, short vowels are typically omitted, leaving readers to reconstruct

Figure 1: An example of a dotted sentence with diacritics (the top line) vs. a dotted sentence without diacritics vs. a dotless sentence (the last line).

phonological and grammatical information from lexical, syntactic, and semantic context. This phenomenon has motivated a major line of research in Natural Language Processing (NLP), namely Arabic diacritization, whose goal is to computationally restore the missing vocalic markers (حركات). Diacritization has consequently become one of the central benchmark tasks in Arabic NLP, with applications ranging from text-to-speech and language education to parsing and machine translation. More broadly, it reflects a fundamental property of abjad writing systems, in which orthographic representation intentionally underencodes pronunciation and delegates part of the interpretive burden to the reader.

Historically, however, the Arabic script exhibited an even deeper level of underspecification. Early Arabic manuscripts were written not only without short vowels, but also without the consonantal dots that distinguish letters such as ب, ت, ث, ن, and ي. The written form therefore represented only a skeletal graphemic structure, traditionally known as the *rasm* (رسم), from which readers inferred both consonantal identity and vocalization through contextual interpretation. Figure 1 contrasts this skele-

tal form with the dotted and diacritized variants familiar to modern readers. Modern Arabic orthography may thus be viewed as a partially specified descendant of a historically more compressed writing system, and Arabic diacritization as one layer of a broader family of orthographic reconstruction problems. A more fundamental and historically prior challenge is the restoration of the consonantal distinctions themselves, a task that we refer to in this work as Arabic rasm restoration or dots restoration.

Despite its linguistic and historical significance, rasm restoration has received comparatively little attention in computational linguistics relative to diacritization, yet the task is arguably a natural extension, and in some respects a deeper generalization, of diacritic restoration. While diacritization operates over a largely disambiguated consonantal sequence, rasm restoration addresses ambiguity at the graphemic level itself, where multiple consonants share the same undotted shape. The inferential burden is therefore substantially greater, requiring the integration of lexical, morphological, syntactic, and semantic information to recover the intended orthographic form. In this sense, rasm restoration provides a particularly revealing testbed for contextual language modeling and probabilistic inference under extreme orthographic ambiguity.

From the standpoint of linguistic theory, both diacritization and rasm restoration can be understood as instances of recovery from underspecified orthography. Rather than encoding all phonological and graphemic distinctions explicitly, the script preserves only the information necessary for reliable interpretation under normal communicative conditions. The recoverability of omitted information depends on the redundancy inherent in natural language: morphology, syntax, lexical constraints, and semantic expectations sharply reduce the space of plausible interpretations, so that reading becomes an inferential process rather than a purely symbolic decoding procedure. This perspective naturally connects Arabic orthography with information theory: the script can be viewed as a form of lossy linguistic encoding, in which predictable information is omitted to reduce representational complexity while preserving communicative efficiency. Letting L denote the underlying linguistic representation and O the observed orthographic form, writing can be modeled as a many-to-one projection $O = f(L)$, and reading as the corresponding probabilistic inverse problem of recovering the most likely L given

O and contextual constraints. Diacritization and rasm restoration are then computational approximations of the same human inferential mechanism, differing primarily in the level of orthographic information omitted from the signal.

Beyond its linguistic and historical interest, rasm restoration carries consequences for several practical fronts in modern NLP. Collapsing distinct dotted letters into a single rasm yields a markedly smaller vocabulary with no appreciable cost to downstream performance (Al-Shaibani and Ahmad, 2025), with implications for embedding-based tasks, tokenizer design, and the broader question of how vocabulary should scale with model size (Tao et al., 2024). Robust dot restoration also enables a natural front end for lightweight OCR on early manuscripts, parchments, and adjacent traditions such as Judeo-Arabic (Gonzalez et al., 2025), with potential extensions to other languages that inherit the Arabic dotting system, including Urdu and Farsi. On the adversarial side, the deliberate removal of dots has begun to be used as a tactic to evade automated content moderation while preserving legibility for human readers, fitting a broader pattern of imperceptible character-level perturbations that degrade NLP systems while remaining intelligible to humans (Boucher et al., 2022).

In this paper, we present *tnqeet*¹, a systematic study of Arabic rasm restoration and position it as a foundational task within this broader family of orthographic reconstruction problems. We compare statistical, neural, self-supervised, and prompt-based approaches in order to evaluate how different paradigms handle orthographic ambiguity at the graphemic level. Figure 2 summarizes the overall framework.

2 Related Work

Alhathloul and Ahmad (2022) introduced the first systematic approach to automatically adding dots to Arabic *rasms* using bidirectional Gated Recurrent Units (BiGRU) following a sequence labeling framework. They evaluated their approach across four diverse datasets, achieving character error rates ranging from 2.0% to 5.5% and dottization error rates between 4.2% and 11.0%, noting that prefix confusion is a primary source of errors. Al-Shaibani and Ahmad (2025) presented the first comprehensive study of dotless Arabic text as an alter-

¹PyPi: <https://pypi.org/project/tnqeet/>,
GitHub: <https://github.com/MagedSaeed/tnqeet>

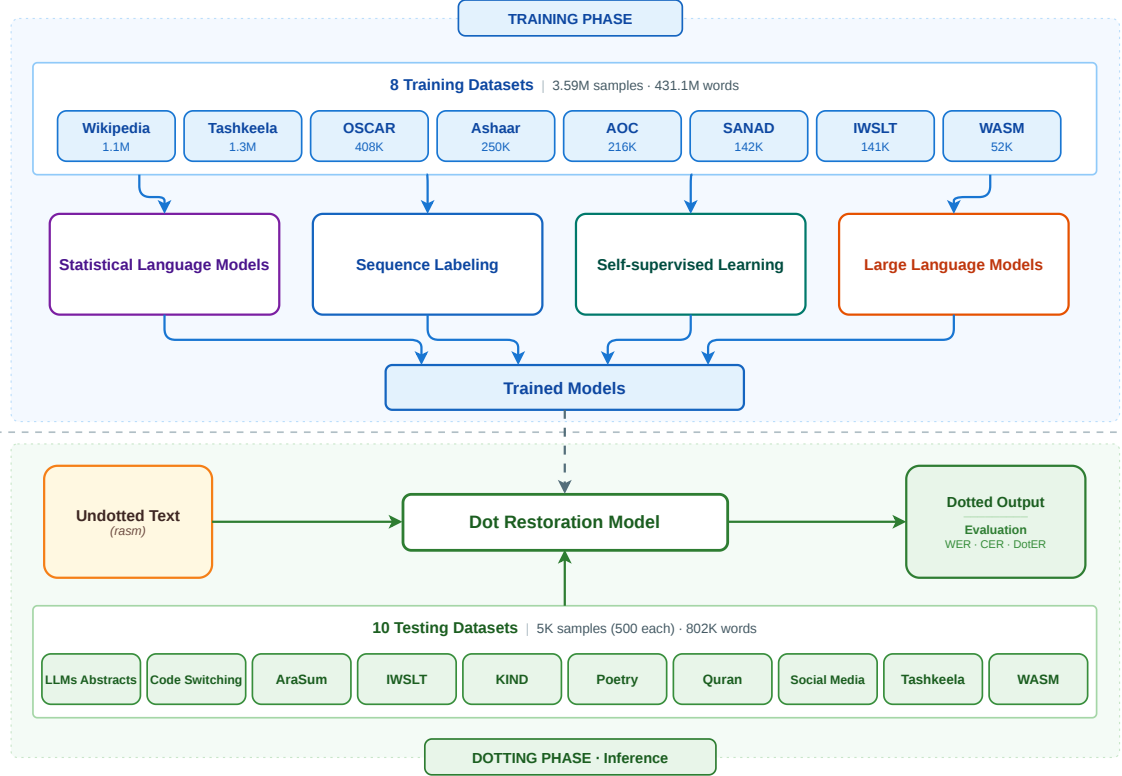


Figure 2: Overview of the *tnqeet* training and evaluation framework.

native representation for various Arabic NLP tasks. Their research demonstrated that removing dots notably reduces vocabulary size while maintaining comparable performance in downstream tasks. Using Zipf’s and Heap’s laws, they showed that dotless text follows similar linguistic patterns to standard Arabic text, achieving vocabulary reductions of up to 50% while maintaining performance across seven different tasks. Alomari et al. (2026) further showed that competitive performance holds even when the dotted-to-dotless mapping is chosen arbitrarily, adding further interesting insights to the task.

Recent work has identified several practical applications for dotless Arabic text processing. Rom and Bar (2021) explored the use of dotless Arabic in social media contexts, where users employ dotless text to circumvent content filtering algorithms, proposing methods to adapt existing language models through tokenizer modification and dot restoration techniques. The historical significance of dotless text has also motivated research into automatic transcription of ancient Arabic manuscripts (Al-hathloul and Ahmad, 2022), representing a direct use case for automatic dottization systems in digital humanities.

Beyond traditional NLP applications, Melhem et al. (2025) demonstrated that graphemic dot-level manipulations can serve as adversarial attacks against Arabic sentiment analysis models and retrieval-augmented generation systems, highlighting the importance of developing robust hybrid dotted and dotless systems. Additionally, recent work on English text simplification provides relevant insights. Al-Shaibani and Ahmad (2023); Alajrami et al. (2023); Alajrami and Aletras (2022) explored consonant-based representations of English text, demonstrating that simplified character representations can maintain effectiveness across NLP tasks while reducing model complexity, sharing conceptual similarities with Arabic dotless representation.

3 Methodology

This section outlines our methodology for Arabic dot restoration (*tnqeet*). We detail the preprocessing pipeline for converting dotted text to its dotless *rasm* form, followed by the construction of the dataset with its splits, followed by modeling methods we explore to restore dots. Figure 2 provides a high-level overview of this methodology.

3.1 Removing Dots

We followed a standard mapping from dotted letters to their dotless variants for letters with more than one form. See Table 2 for the equivalent dotless letter of each dotted letter. Except for four letters, the dotless variants coincide with letters already present in the common Arabic character set and require no special Unicode lookup. Only four letters need special Unicode codepoints for their dotless forms: ب, ن, ف, and ق. Before dot removal, we apply a superficial Unicode normalization step that drops unprintable characters and maps rare Unicode variants to their standard counterparts. Unlike Al-Shaibani and Ahmad (2025), we do not implement any special handling for the cases of ن or ي when they appear in the middle of a word.

3.2 Datasets

To train and evaluate dot restoration approaches across distributions, we compiled corpora from diverse Arabic domains, including several held out from training to probe generalization. Across all splits, we retain only samples of at least 10 words to maintain useful context. Tables 3 and 4 in Appendix B report full per-source statistics.

The training set comprises 3.59M samples and 431.1M words drawn from eight sources: Arabic Wikipedia, Tashkeela for religious and classical texts, poetry, news, translation and speech, web text, and dialectal social media. For validation, we constructed three stratified sets by sampling from the first 10K examples of each training source: a main set of 150 examples per source for development and hyperparameter tuning, a smaller LLM-evaluation set of 15 examples per source, given the cost of flagship-model experiments, and a non-overlapping LLM few-shot set of the same size for in-context examples. All validation sets were shuffled with a fixed random seed for reproducibility.

The test set contains 5K samples (500 per source) across variety of domains, totalling 801K words with an average of 160.2 words per sample. It spans academic abstracts, code-switched Ar/En text, news summaries, translation and speech, conversational Q&A, poetry, religious and classical text, and social media. Three of these domains, LLM-generated academic abstracts, conversational Q&A, and code-switched domains, are deliberately absent from the training set so we can probe generalization to unseen distributions.

3.3 Evaluation Metrics

We evaluate dot restoration performance using three complementary metrics. **Word Error Rate (WER)** measures the percentage of words that differ between the reference and predicted text. **Character Error Rate (CER)** calculates the character edit distance between reference and predicted sequences. **Dotting Error Rate (DotER)**² (Alhathloul and Ahmad, 2022) specifically measures errors in ambiguous *rasm* characters that require dot disambiguation, focusing on the core task of contextual dot restoration while ignoring characters that remain unchanged from dotless to dotted forms.

3.4 Models

We evaluate four approaches for Arabic dot restoration (*tnqeet*), arranged along a spectrum of training objective and supervision: (i) classical statistical models that score sequences with n -gram counts, (ii) neural sequence labeling models trained from scratch (BiLSTMs and Transformers), (iii) self-supervised pre-trained character-level fine-tuned models (CANINE), and (iv) large language models prompted via few-shot in-context learning. Each approach represents a different paradigm for the contextual disambiguation of dotless Arabic text restoration. This combination allows for various deployment options where sequence labeling and self-supervised learning are suitable for efficiency, while statistical methods are the best for compute-constrained environments (environments that does not require GPUs), and LLMs are suitable for cloud-based scenarios with suitability to general language understanding tasks.

3.4.1 Statistical N-grams Models

Classical n -gram language models predict each token based on the previous $n - 1$ tokens, approximating the probability of a sequence $w_1, w_2, \dots, w_m$ as in Equation 1. We implement models ranging from 2-grams to 10-grams using KenLM (Heafield, 2011) with modified Kneser-Ney smoothing (Kneser and Ney, 1995). For n -grams of order 3 and higher, we prune singleton n -grams to reduce model size. Models are trained on standard Arabic dotted text at the character level, with whitespace replaced by <SPACE> tokens. For inference, the n -gram model provides contextual probability scores while beam search explores the

²Introduced as *DoER* in (Alhathloul and Ahmad, 2022)

K -best dotting alternatives. We use beam sizes ranging from 1 to 100, as described in Algorithm 1. We create a reverse mapping from each *rasm* to its possible dotted forms (Table 2), passing unambiguous characters (e.g., ل, م, ء) unchanged.

3.4.2 Sequence Labeling Models

We frame the dotting problem as a sequence labeling task, where each dotless character is mapped to its corresponding dotted character. We explore two neural architectures trained from scratch: bidirectional Long Short-Term Memory networks (BiLSTMs) and Transformer encoders. We include both because they offer complementary objectives for character-level Arabic processing: BiLSTMs encode the input recurrently with a hidden state that favors local, sequentially aggregated context, while Transformers use self-attention to model interactions across the full sequence in parallel, which can be advantageous when disambiguating a *rasm* requires evidence from words several positions away. Comparing both lets us isolate the contribution of architecture, holding training data, tokenization, and optimization setup fixed. We ablate 1 to 6 layers for BiLSTMs and 3, 6, 9, 12 layers for Transformers. Section C.2 reports the full training and inference setup.

3.4.3 Self-Supervised Pre-trained Models

Beyond training from scratch, we investigate whether transferring representations learned through large-scale self-supervised pre-training improves dot restoration. Since the task is inherently character-level, we select a pre-trained model whose input granularity matches the task. We fine-tune CANINE (Clark et al., 2022), a tokenization-free pre-trained encoder that operates directly on Unicode codepoints, letting us reuse pre-trained representations without reformulating the problem. At inference, predictions are applied only at ambiguous *rasm* positions, with unambiguous characters retrieved from the input unchanged, to stay consistent with the inference setup used for the other approaches. Section C.3 reports the fine-tuning details.

3.4.4 Large Language Models

We evaluate multiple state-of-the-art LLMs using the DSPy (Khattab et al., 2023) framework for structured prompting and few-shot learning. We evaluate 11 different LLMs via OpenRouter³.

³<https://openrouter.ai/>

Our model selection includes closed-source models (Claude Sonnet 4, Claude Haiku 3.5, GPT-4o, GPT-4o Mini, Gemini 2.5 Flash Preview, Gemini 2.5 Flash Lite) and open-source alternatives (Gemma 3 27B, Llama 3.3 70B, Qwen 3 235B, Qwen 3 32B), covering a range of model sizes from compact variants (Haiku, Mini, Lite) to flagship large models (Qwen 3 235B, Claude Sonnet, DeepSeek). We implement two prompting methods. The first directly asks the model to return the dotted text given the dotless input. The second additionally includes the dotted-to-dotless mapping we applied, hoping this improves restoration performance. We experiment with few-shot settings of 0, 1, 3, 5, 8, and 10 examples of dotless-to-dotted text pairs. For evaluation consistency, we truncate model outputs to match the length of the original input text when computing metrics, ensuring fair comparison across all approaches.

4 Results and Analysis

In this section, we present results on both validation and test sets. Validation results informed our hyperparameter selection and model configuration choices for the final test set evaluation.

Validation Dataset Results On the validation set, n -grams start to perform competitively ($\approx 6\%$ WER) at orders 5–8 with beam sizes starting from 10; gains plateau beyond order 8 while model size grows exponentially (a factor of 3.56 per order, reaching 1.3 GB at order 10 versus 0.01 GB at order 3), motivating our choice of 6-, and 8-gram models for the test sets. We also added 3 and 4 grams to report simpler configurations. BiLSTMs reach this performance (5.35 WER at 6 layers, starting to saturate at 3 layers) while offering substantially faster, parallel inference than the autoregressive alternatives. Transformers push this further, reaching 4.03 WER at 12 layers with gains saturating beyond 9 layers. CANINE performs slightly below transformers, albeit at a higher per-character processing cost. In contrast, the 11 evaluated LLMs perform notably worse, even the flagship models (Claude Sonnet 4, Gemini 2.5 Flash Preview) reach only 15%–21% WER, with wide variance (15%–75%), inconsistent gains beyond 3–5 shots, and minimal sensitivity to prompt style. The detailed discussion is deferred to Appendix D.

Test Dataset Results Table 1 presents the comprehensive evaluation of all approaches on the test

Table 1: Performance comparison of all approaches on the test dataset. Best results are shown in bold. WER, CER, and DotER are reported as percentages. Inference cost is mean milliseconds per source character for models other than LLMs, and mean prompt/completion tokens per example for LLMs.

Method	Setting	WER (%)	CER (%)	DotER (%)	Inference cost
3-gram	beam=15	30.63	6.43	7.52	0.0540 ms/char
4-gram	beam=30	19.66	3.94	4.58	0.1070 ms/char
6-gram	beam=50	13.76	2.72	3.14	0.1848 ms/char
8-gram	beam=60	13.32	2.59	2.99	0.2239 ms/char
BiLSTM	2 layers	9.21	1.77	2.06	0.0745 ms/char
BiLSTM	4 layers	8.49	1.62	1.87	0.1345 ms/char
BiLSTM	6 layers	8.43	1.59	1.84	0.2086 ms/char
Transformer	3 layers	8.35	1.64	1.91	0.0214 ms/char
Transformer	6 layers	7.08	1.39	1.62	0.0294 ms/char
Transformer	9 layers	7.01	1.36	1.58	0.0369 ms/char
Transformer	12 layers	6.97	1.33	1.55	0.0445 ms/char
CANINE	canine-c	7.66	1.53	1.78	0.0839 ms/char
CANINE	canine-s	7.48	1.48	1.72	0.0965 ms/char
Claude Sonnet 4	8-shot	20.42	4.86	5.59	8,096 / 708 tok
Gemini 2.5 Flash Preview	8-shot	23.59	6.05	6.87	5,074 / 416 tok
GPT-4o	8-shot	42.73	14.81	17.05	5,237 / 374 tok

dataset, with configurations informed by validation set performance. For LLMs, we evaluated only the top three performers (Claude Sonnet 4, Gemini 2.5 Flash Preview, and GPT-4o) from our initial 11-model validation experiments. Note that Haiku was better than GPT-4o in the validation results, but we choose the latter considering provider variability. We used default prompting given the minimal impact of prompt engineering variations observed during validation. On the test set, the ranking observed during validation largely holds, though absolute WERs rise across the board. The 12-layer Transformer remains the strongest, and its 3-layer counterpart is the cheapest overall while still matching the deepest BiLSTM. CANINE models score between the LSTMs and transformers at roughly twice the per-character cost of the transformer. Statistical n-grams require higher orders to be competitive with sequence labeling methods while incurring the highest inference cost. In contrast, LLMs lag far behind with the best of the three evaluated, Claude Sonnet 4 with 8 shots, reaching only 20.42 WER, roughly on par with a 4-gram model, while GPT-4o trails at 42.73 WER. This confirms that current LLMs poorly support dotless Arabic despite the simplicity of the task.

Comparison with the BiGRU baseline We benchmark our dotters against the original recurrent dottization study of [Alhathloul and Ahmad \(2022\)](#) on the Arabic Treebank (ATB; 3,538 samples), the only portion of their four-corpus bench-

mark that our training data does not already cover. Their gold additionally normalizes $\tilde{\text{ا}}$ to ا and $\tilde{\text{و}}$ to و , letters that our models instead restore, so we apply the same normalization to both sides before scoring rather than penalizing our models for restoring them. All four of our methods outperform their BiGRU, from the 6-layer BiLSTM down to CANINE, against their reported scores. Notably, in this setup, CANINE drops relative to its ranking in Table 1, where it outperforms every n-gram model. We investigated further and found out that about 89% of the n-gram’s raw character errors fall on ا and و where they vanish once both sides are normalized, whereas CANINE, despite fewer raw errors overall, distributes its mistakes on other letters. Appendix E details the setup and the full results.

Per-domain difficulty Figure 3 decomposes each model’s WER across the 10 test domains, where every domain contributes 500 samples so that a segment reflects how hard a domain is within the context of this task. For the four trained methods, the hardest domains are the KIND dataset (the conversational Q/A dataset) and the code-switched Arabic/English text, both dialectal and both held out from training. The LLM-generated academic abstracts, in contrast, stay among the easiest domains for every method despite also being held out, since their register is MSA-like and is machine-generated content. Poetry follows for the

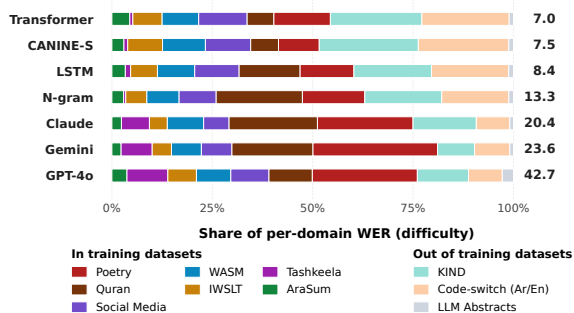


Figure 3: Per-domain attribution of models' WERs on the test set (rows sum to 100%); the right-hand number is overall WER (%).

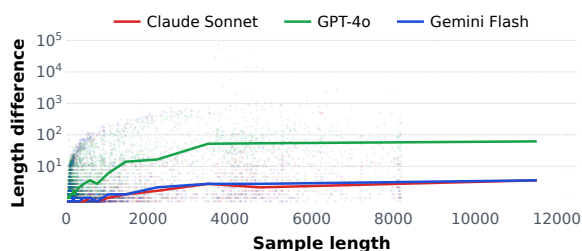


Figure 4: Absolute character-length difference between the LLM output and the reference text, as a function of input length; the bold lines are binned medians over 25 input-length bins. The y -axis is symlog (linear below 10).

trained methods, with the Quran close behind for the n-gram and the BiLSTM, which we attribute to the unique structure and language that set these datasets apart as classical language. The LLMs order the domains by register instead, finding MSA-like text easiest and the classical registers of poetry, the Quran, and Tashkeela hardest, the last of which the trained methods address effectively.

LLMs hallucination with a fixed output Because LLMs operate autoregressively rather than as sequence-labeling models, guaranteeing exact preservation of the input sequence with only dot insertions is challenging, particularly in light of the known hallucination behavior of LLMs (Alansari and Luqman, 2026). This behavior was analyzed on the test set using the selected LLMs. Figure 4 shows the scatter plot of input-output differences in correlation with sample length in characters. Note that diacritics are stripped for convenience. As the figure shows, the sample length affects the prediction robustness, although plateaus at a certain length. Interestingly, out of the 5k samples in the test set, only 48%, 47%, and 15% of outputs from Claude Sonnet 4, Gemini 2.5 Flash Preview,

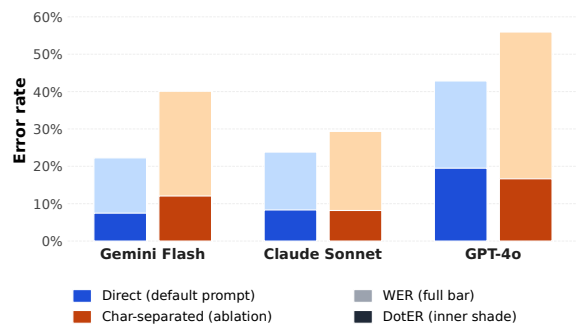


Figure 5: Zero-shot WER and DotER on the validation set for three LLMs with character-separated inputs vs. direct default prompting.

and GPT-4o, respectively, exactly match the input length.

LLMs ablation with Character-separated prompting

Because BPE tokenizers merge multiple Arabic characters into a single token, LLMs may not process individual glyphs explicitly when predicting dots, which could partly explain their weaker performance on this task. This is also observed on diacritization, where fully diacritized text shows degradation due to inefficient subword tokenization (Inoue et al., 2026). Alyafeai et al. (2023b) showed that different tokenizations work best for different tasks. Accordingly, we tested whether enforcing one-character-per-token helps by re-evaluating the three top LLMs zero-shot on the validation set with both input and output space-separated at the character level and word boundaries marked with |. Figure 5 compares this format against the default direct prompting. WER degrades across all three models, yet DotER behaves differently: a marginal reduction for Claude, a drop for GPT-4o, and only a modest rise for Gemini. A closer look reveals that the format does improve per-glyph dottization, but errors spread across more words; DotER, being essentially per character, was able to capture this improvement better than WER.

Dotless Arabic in the current LLMs landscape

The persistent LLM lag prompts a closer look at how current models support dotless Arabic at the tokenization level. Figure 6 reports the fraction of vocabulary tokens containing at least one rasm character across eighteen different LLMs tokenizers, grouped by the number of distinct rasm characters present in the vocabulary. Here we only consider the four rare rasm letters: (ب, ن, ف, ق), since

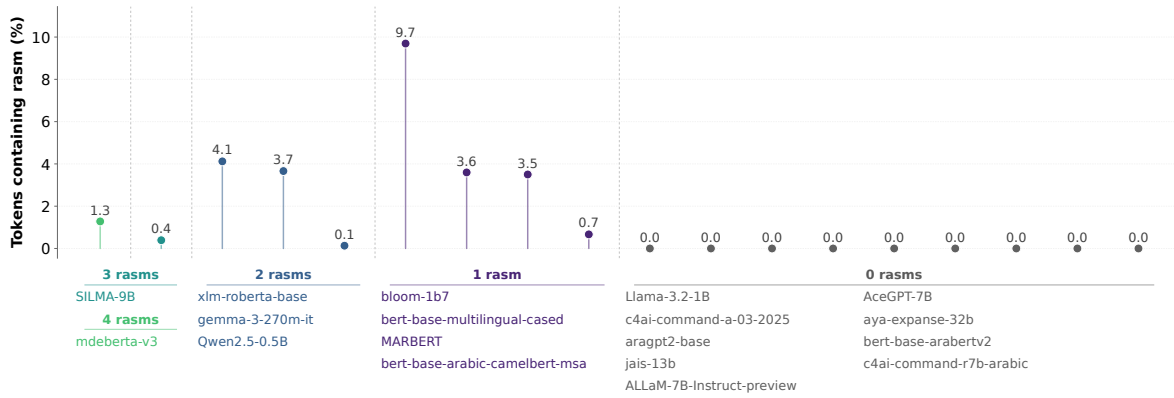


Figure 6: Rare rasms presence across 18 LLM tokenizers.

the remaining dotless rasm forms coincide with letters already in standard Arabic character set. The figure shows that LLMs across different training objectives lag in supporting dotless text, with the maximum coverage being roughly 10% of tokens for BLOOM, while many of these models do not contain a single rasm letter in their tokenizer. From a tokenization-compression perspective, following (Al-Shaibani and Ahmad, 2025; Alyafeai et al., 2023b) insights, we further quantify the vocabulary reduction obtained by undotting each tokenizer’s Arabic tokens and merging the duplicates, depicted in Figure 7. Across fifteen tokenizers, undotting yields a consistent reduction of between 11.3% and 21.1%, with the larger values on Arabic-first tokenizers. These insights motivate further exploration of employing dotless text in downstream tasks such as embedding-based applications, where this reduction is valuable.

Per-position error attribution We examine where dotting errors concentrate along the letter’s relative position in its words. For a word with length ℓ , the relative letter position i is calculated as: $i/(\ell-1)$ for $\ell > 1$ and $i \geq 0$. Figure 8 illustrates this analysis. Middle positions are easiest, but word initials and finals are consistently the hardest among all methods. The bottom density panel provides another view where some middle positions (around 0.5) are harder to predict than others, but word starts and ends are the hardest overall, regardless of the dotting method utilized.

How much context does dotting need? To study how much prior context dot restoration actually requires, we feed each model only the first n words of every validation example, score the prediction against those same n words, and vary n from 1

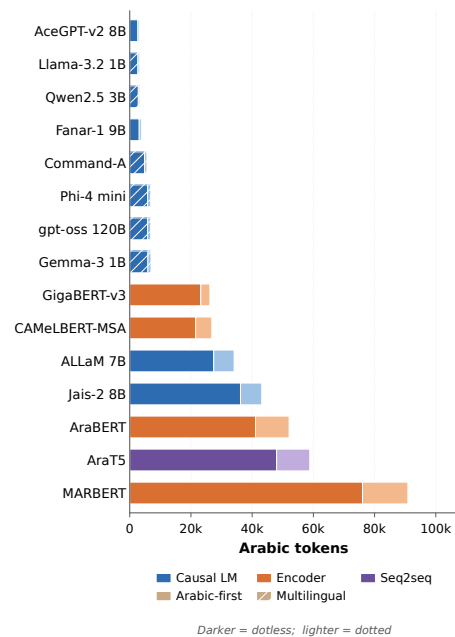


Figure 7: Vocabulary reduction by undotting Arabic tokens. Right-side values give the per-tokenizer reduction after undotting; the mean across all models is 16.23%.

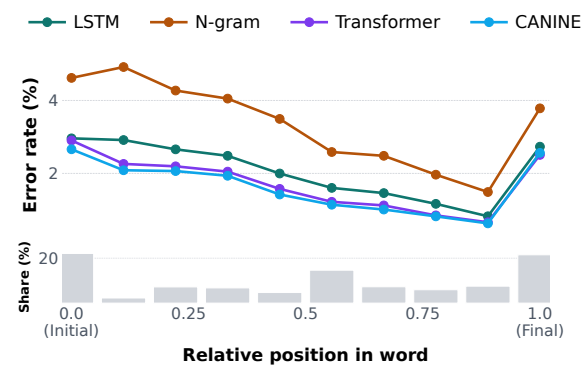


Figure 8: Per-letter error rate against the letter’s relative position in its word; the bottom panel shows the error share of dotless letters positions per bin.

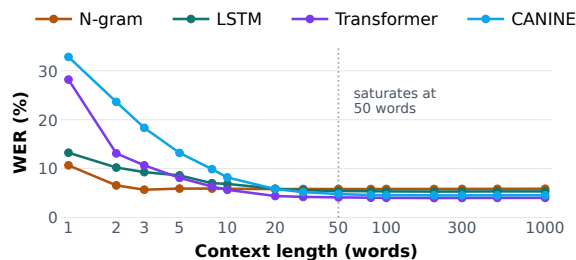


Figure 9: Validation WER as a function of the number of context words available to each model;

to 1000 using the best configuration of each non-LLM method. Figure 9 shows the resulting curves on the WER metric. All four methods improve to roughly ten words then start to converge until they plateau around 50 words. Thus, context beyond a couple of sentences has little contribution to the model WER. The methods differ considerably in how much context they need to perform effectively. The 8-gram reaches its floor after only three words and stays almost flat, which makes it the strongest method up to eight words and the weakest beyond thirty, whereas CANINE and the Transformer start at 32.83 and 28.20 WER on a single word yet settle well below it. This suggests that dotting is largely affected by the nearby context. Additionally, this brings useful insights on utilizing n-grams for dotless text in short inputs.

5 Conclusion

This work presents a comprehensive benchmark of four families of approaches for Arabic dot restoration (*tnqeet*): statistical n-gram language models, neural sequence labelers trained from scratch (BiLSTMs and Transformers), self-supervised character-level encoders (CANINE), and prompted large language models. On an aggregated test set spanning a variety of domains, Transformers achieve the best performance at the lowest per-character cost among neural methods, with CANINE next and BiLSTMs trailing, though CANINE incurs roughly twice the per-character inference cost of the deepest Transformer. Classical n-grams remain competitive at higher orders despite their computational simplicity. Notably for LLMs, despite their remarkable capabilities on general language understanding tasks, including flagship models, they lag substantially on this seemingly shallow character-level task, with the best of those evaluated (Claude Sonnet 4, 8-shot) roughly matching a 4-gram baseline. Tokenizer analysis offers a plau-

sible explanation suggesting that dotless Arabic is severely under-represented in current LLM vocabularies, with most models containing none of the four rare rasm characters. Per-domain attribution further shows that text distribution accounts for most residual errors across methods, regardless of whether the domain is in training or not, suggesting that diverse training coverage matters more than domain coverage. Taken together, these results show character-aware sequence labelers as the most efficient method for this task. Future work can consider dotless text for downstream tasks such as OCR, information retrieval, semantic similarity, and the broader role of dotless representations in vocabulary-efficient Arabic language modeling.

Limitations

In this work, our setup assumes fully dotless input, a deliberate choice that yields a controlled benchmark and characterizes the upper bound of the task’s difficulty, since complete dot removal maximizes graphemic ambiguity. Partially dotless text is nonetheless more common in practice, particularly on social media, and evaluating on it is a natural next step, though it requires parallel dotless–dotted corpora that are hard to obtain at scale. We also studied only the standard rasm mapping; further variants, such as merging medial ن and ي into the dotless ب family (Al-Shaibani and Ahmad, 2025), are left for future work, as this setup complicates the task definition. While the dataset we compiled is sufficiently large, covering more diverse text would add value, as supported by our analysis. On the LLM side, we considered two prompt engineering methods and left more advanced ones (such as chain-of-thought, role playing, etc.) aside, since detailed prompting provided only marginal improvements. We leave the evaluation of the dot restoration task in downstream end-to-end scenarios (OCR, retrieval, similarity) as future work, along with multilingual extension experiments to related scripts such as Urdu and Farsi.

Acknowledgements

The authors would like to thank King Fahd University of Petroleum and Minerals (KFUPM) and SDAIA–KFUPM Joint Research Center for AI for supporting this work.

References

- Wafa Al Ghallabi, Ritesh Thawkar, Sara Ghaboura, Ketan Pravin More, Omkar Thawakar, Hisham Cholakkal, Salman Khan, and Rao Muhammad Anwer. 2025. Fann or flop: A multigenre, multiera benchmark for arabic poetry understanding in llms. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20235–20255.
- Maged Al-Shaibani and Irfan Ahmad. 2023. Consonant is all you need: A compact representation of english text for efficient nlp. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11578–11588.
- Maged S Al-Shaibani and Irfan Ahmad. 2025. Dotless arabic text for natural language processing. *Computational Linguistics*, 51(2):557–598.
- Maged S Al-Shaibani and Moataz Ahmed. 2025. Arabic machine-generated text detection: Stylometric analysis and cross-model evaluation. *Expert Systems with Applications*, page 130644.
- Maged S Al-Shaibani, Hamzah Luqman, Abdulaziz S Al-Ghofaily, and Abdullatif A Al-Najim. 2024. Wasm: A dataset for hashtag recommendation for arabic tweets. *Arabian Journal for Science and Engineering*, 49(9):12131–12145.
- Ahmed Alajrami and Nikolaos Aletras. 2022. How does the pre-training objective affect what large language models learn about linguistic properties? In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 131–147.
- Ahmed Alajrami, Katerina Margatina, and Nikolaos Aletras. 2023. Understanding the role of input token characters in language models: How does information loss affect performance? In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9085–9108.
- Aisha Alansari and Hamzah Luqman. 2026. [Large language models hallucination: A comprehensive survey](#). *Computer Science Review*, 61:100970.
- Zainab Alhathloul and Irfan Ahmad. 2022. Automatic dottization of arabic text (rasms) using deep recurrent neural networks. *Pattern Recognition Letters*, 162:47–55.
- Dorieh Alomari, Irfan Ahmad, and Maged S Al-shaibani. 2026. Character iconicity vs. arbitrariness: An arabic nlp perspective. *arXiv preprint arXiv:2608.02935*.
- Saied Alshahrani, Norah Alshahrani, Soumyabrata Dey, and Jeanna Matthews. 2023. [Performance Implications of Using Unrepresentative Corpora in Arabic Natural Language Processing](#). In *Proceedings of the The First Arabic Natural Language Processing Conference (ArabicNLP 2023)*, pages 218–231, Singapore (Hybrid). Association for Computational Linguistics.
- Zaid Alyafeai, Maged S. Al-Shaibani, and Moataz Ahmad. 2023a. [Ashaar: Automatic analysis and generation of arabic poetry using deep learning approaches](#). *ArXiv*, abs/2307.06218.
- Zaid Alyafeai, Maged S Al-shaibani, Mustafa Ghaleb, and Irfan Ahmad. 2023b. Evaluating various tokenizers for arabic text classification: Z. alyafeai et al. *Neural Processing Letters*, 55(3):2911–2933.
- Nicholas Boucher, Ilia Shumailov, Ross Anderson, and Nicolas Papernot. 2022. [Bad characters: Imperceptible NLP attacks](#). In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1987–2004. IEEE.
- Mauro Cettolo, Marcello Federico, Luisa Bentivogli, Jan Niehues, Sebastian Stüker, Katsuhito Sudoh, Koichiro Yoshino, and Christian Federmann. 2017. [Overview of the IWSLT 2017 evaluation campaign](#). In *Proceedings of the 14th International Conference on Spoken Language Translation*, pages 2–14, Tokyo, Japan. International Workshop on Spoken Language Translation.
- Jonathan H Clark, Dan Garrette, Iulia Turc, and John Wieting. 2022. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91.
- Omar Einea, Ashraf Elnagar, and Ridhwan Al Debsi. 2019. Sanad: Single-label arabic news articles dataset for automatic text categorization. *Data in brief*, 25:104076.
- Juan Moreno Gonzalez, Bashar Alhafni, and Nizar Habash. 2025. A tale of two scripts: Transliteration and post-correction for Judeo-Arabic. *arXiv preprint arXiv:2507.04746*.
- Kenneth Heafield. 2011. [Kenlm: Faster and smaller language model queries](#). In *WMT@EMNLP*.
- Go Inoue, Bashar Alhafni, Nizar Habash, and Timothy Baldwin. 2026. Do diacritics matter? evaluating the impact of arabic diacritics on tokenization and llm benchmarks. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 426–442.
- Mram Kahla, Zijian Győző Yang, and Attila Novák. 2021. [Cross-lingual fine-tuning for abstractive Arabic text summarization](#). In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, pages 655–663, Held Online. INCOMA Ltd.
- O. Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023. [Dspy: Compiling declarative language model calls into self-improving pipelines](#). *ArXiv*, abs/2310.03714.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184. IEEE.

Abdullah Melhem, Ahmed Aleroud, and Craig Douglas Albert. 2025. GraphRAG-based NLP at risk: Graphemic dot-level adversarial attack on Arabic sentiment and LLM retrieval-augmented models. In *Proceedings of the 2025 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*.

Pedro Javier Ortiz Su’arez, Laurent Romary, and Benoit Sagot. 2020. [A monolingual approach to contextualized word embeddings for mid-resource languages](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1703–1714, Online. Association for Computational Linguistics.

Pedro Javier Ortiz Su’arez, Benoit Sagot, and Laurent Romary. 2019. [Asynchronous pipelines for processing huge corpora on medium to low resource infrastructures](#). Proceedings of the Workshop on Challenges in the Management of Large Corpora (CMLC-7) 2019. Cardiff, 22nd July 2019, pages 9 – 16, Mannheim. Leibniz-Institut f"ur Deutsche Sprache.

Mohamed Rashad. 2024. [arabic-english-code-switching](#).

Aviad Rom and Kfir Bar. 2021. Supporting undotted arabic with pre-trained language models. In *Proceedings of the 4th International Conference on Natural Language and Speech Processing (ICNLSP 2021)*, pages 89–94.

Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muenighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. 2024. [Scaling laws with vocabulary: Larger models deserve larger vocabularies](#). In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*.

Asma Yamani, Raghad Alziyady, Reem AlYami, Salma Albelali, Leina Albelali, Jawharah Almulhim, Amjad Alsulami, Motaz Alfarraj, and Rabeah Al-Zaidy. 2024. [The KINd dataset: A social collaboration approach for nuanced dialect data collection](#). In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 32–43, St. Julian’s, Malta. Association for Computational Linguistics.

Omar Zaidan and Chris Callison-Burch. 2011. The arabic online commentary dataset: an annotated dataset of informal arabic with high dialectal content. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 37–41.

Taha Zerrouki and Amar Balla. 2017. Tashkeela: Novel corpus of arabic vocalized texts, data for auto-diacritization systems. *Data in brief*, 11:147.

A Dotless Mapping

Table 2 illustrates the mapping from dotted to dotless letters followed in this study.

Table 2: Mapping of dotted Arabic letters to their undotted rasm equivalents

Rasm	Dotted	Rasm	Dotted
ا	ا، ا، آ	ص	ص، ض
ب	ب، ت، ث	ط	ط، ظ
ح	ح، ج، خ	ع	ع، غ
د	د، ذ	ف	ف
ر	ر، ز	ق	ق
س	س، ش	ك	ك
ن	ن	ل	ل
ه	ه، ة	م	م
و	و، ؤ	ء	ء
ي	ي، ي، ئ		

B Training and Testing Datasets Details

This section presents the datasets details utilized in this study. Note that we implemented some rules for certain datasets to maintain domain diversity. For the Tashkeela corpus, we used it both for diacritized and classical-based Arabic content. For training, we split the dataset into two halves: for the first half, we selected it as classical Arabic text; for the second half, we selected it as diacritized text. We reversed this process when selecting test samples. For the LLM Abstracts dataset ([Al-Shaibani and Ahmed, 2025](#)), we specifically extracted LLM-generated content from multiple language models (Allam, Jais, Llama, and OpenAI) across three different generation tasks (by polishing, from title, and from title and content), excluding human-written abstracts to focus evaluation on machine-generated text patterns.

C Further modeling details

C.1 n-grams inferencing

N-gram models prediction according to the following equation:

Table 3: Training dataset statistics. The training dataset contains over 3.59 million samples spanning diverse domains and language varieties.

Dataset Name	Domain	Language	Samples	Total Words	Avg Words
Annotated AOC (Zaidan and Callison-Burch, 2011)	Social Media	Dialectal	216K	7.1M	32.9
Arabic Wikipedia (Alshahrani et al., 2023)	Wikipedia	MSA	1.1M	258.7M	237.8
Ashaar (Alyafeai et al., 2023a)	Poetry/ Literature	Classical Arabic	250K	35.6M	142.3
IWSLT (Cettolo et al., 2017)	Translation/ Speech	MSA	141K	2.8M	19.6
OSCAR (small) (Ortiz Su'arez et al., 2020; Ortiz Su'arez et al., 2019)	Web Text	MSA/ Dialectical	408K	19.0M	46.5
SANAD (Einea et al., 2019)	News/Media	MSA	142K	35.8M	252.7
Tashkeela (Zerrouki and Balla, 2017)	Religious/ Classical	Classical Arabic	1.3M	71.0M	55.0
WASM (Al-Shaibani et al., 2024)	Social Media	Dialectal	52K	1.1M	20.2
TOTAL			3.59M	431.1M	

Table 4: Testing dataset statistics. The testing dataset provides 500 balanced samples across different Arabic text types for each domain.

Dataset Name	Domain	Language	Samples	Total Words	Avg Words
LLMs Abstracts (Al-Shaibani and Ahmed, 2025)	Academic	MSA	500	41K	82.2
ar-en code switching (Rashad, 2024)	Code Switching	Dialectal	500	11K	22.2
AraSum (Kahla et al., 2021)	News/Media	MSA	500	185K	370.9
IWSLT 2017 (Cettolo et al., 2017)	Translation/ Speech	MSA	500	10K	19.1
KIND (Yamani et al., 2024)	Conversational/ Q&A	Dialectal	500	8K	15.6
poetry (Al Ghallabi et al., 2025)	Poetry/ Literature	Classical Arabic	500	69K	138.6
Quran	Religious/ Classical	Classical Arabic	500	9K	18.2
Social Media (Al-Shaibani and Ahmed, 2025)	Social Media	MSA/ Dialectal	500	431K	861.8
Tashkeela (Zerrouki and Balla, 2017)	Religious/ Classical	Classical Arabic	500	27K	54.7
WASM (Al-Shaibani et al., 2024)	Social Media	Dialectal	500	10K	19.6
TOTAL			5K	802K	

$$P(w_1, \dots, w_m) \approx \prod_{i=1}^m P(w_i \mid w_{i-n+1}, \dots, w_{i-1}) \quad (1)$$

Algorithm 1 N-Gram Beam Search for Arabic Dotting

1. **Input:** Dotless text D , beam size k , n-gram model M , Rasm-to-letters map R
 2. **Output:** Best dotted sequence S^*
 3. $T = \text{Tokenize}(D)$ // split into chars; each whitespace becomes a single `<SPACE>` token
 4. $s_0 = M.\text{BeginSentenceState}()$ // base scoring on `<s>`
 5. Initialize beam $B = \{(0.0, \emptyset, s_0)\}$ // (cumulative log-prob, sequence, LM state)
 6. **for** each token d_i in T **do**
 - (a) candidates = $R[d_i]$ **if** $d_i \in R$ **else** $[d_i]$
 - (b) $B_{\text{new}} = \emptyset$
 - (c) **for** each (score, seq, state) in B **do**
 - i. **for** each *cand* in candidates **do**
 - A. $(\delta, \text{state}') = M.\text{BaseScore}(\text{state}, \text{cand})$
// $\delta = \log P(\text{cand} \mid \text{state})$
 - B. $\text{new_score} = \text{score} + \delta$
 - C. $B_{\text{new}} += (\text{new_score}, \text{seq} + [\text{cand}], \text{state}')$
 - (d) **if** $|B_{\text{new}}| > k$ **then** $B = \text{TopK}(B_{\text{new}}, k)$ **else** $B = B_{\text{new}}$
 7. $S^* = \arg \max_{(s, \text{seq}, \cdot) \in B} s$
 8. **return** $\text{Detokenize}(S^*)$ // join chars; replace `<SPACE>` with ' '
-

We followed algorithm 1 to predict new characters based on a given context of previous characters.

C.2 Sequence labeling modeling details

BiLSTM models. The BiLSTM model consists of an embedding layer of dimension 512, followed by BiLSTM layers with configurable depth, hidden size 512, and dropout rate 0.33, followed by a linear output layer. We experiment with 1, ..., 6, LSTM layers to investigate the effect of model depth on restoration accuracy. We chose BiLSTMs

as they can see both left and right context when making predictions for each character position. We optimize with Adam at a learning rate of 0.001 and a linear-decay schedule, with gradients clipped at 1.0.

Transformer models. We use the standard PyTorch Transformer encoder (`nn.TransformerEncoder`) with a model dimension of 512, 8 attention heads, a feedforward dimension of 2048, dropout of 0.1, and sinusoidal positional encodings, followed by a linear output projection over the tokenizer vocabulary. We vary the number of encoder layers across $\{3, 6, 9, 12\}$ to study the effect of depth on restoration accuracy. We optimize with AdamW ($\beta = (0.9, 0.98)$, $\varepsilon = 10^{-6}$, weight decay 0.01) with an inverse-square-root learning rate schedule, a peak learning rate of 3×10^{-4} , and 4000 warmup steps; gradients are clipped at 0.5, and training uses bfloat16 mixed precision.

Training pipeline. Both BiLSTM and Transformer models use the same character-level tokenizer and are trained with cross-entropy loss. We apply the same early-stopping policy: a patience of 4 validation evaluations, with 4 evaluations per epoch. Early-stopping validations are computed on a 5% held-out split of the training set, distinct from the validation set used for model selection. The maximum sequence length is 1024 characters; longer texts are split at word boundaries and reconciled after each part has had its dots restored.

C.3 CANINE fine-tuning details

We start from the `google/canine-c` checkpoint and add a token-classification head consisting of a dropout layer (rate 0.1) followed by a linear projection over a label vocabulary covering all dotted Arabic characters, ASCII characters, and common Arabic punctuation. The CANINE backbone operates at the character level while internally simulating the subword-tokenization process for efficiency. The model is fine-tuned end-to-end with cross-entropy loss using AdamW (learning rate 3×10^{-5} , weight decay 0.01) and a linear warmup-and-decay schedule with a 10% warmup ratio. Training uses the same early-stopping criteria as the from-scratch models.

C.4 LLM Prompting Signatures

We implemented two prompting approaches using DSPy signatures for LLM-based dot restoration.

Listing 1 shows the direct prompting signature with minimal context where listing 2 shows the actual prompt sent to the LLM, while Listing 3 presents the detailed signature that includes explicit character mappings and 4 shows the actual prompt sent to the LLM.

```

1 class ArabicDottingSignature(dspy.Signature):
2     dotless_text = dspy.InputField(
3         desc="Arabic text without dots (Rasm)"
4     )
5     dotted_text = dspy.OutputField(
6         desc="Text with proper dots added"
7     )

```

Code 1: Direct prompting signature with minimal context

```

1 System message:
2
3 Your input fields are:
4 1. `dotless_text` (str): Arabic text without dots (Rasm)
5 Your output fields are:
6 1. `dotted_text` (str): Text with proper dots added
7 All interactions will be structured in the following way, with the appropriate values filled in.
8
9 [[ ## dotless_text ## ]]
10 {dotless_text}
11
12 [[ ## dotted_text ## ]]
13 {dotted_text}
14
15 [[ ## completed ## ]]
16 In adhering to this structure, your objective is:
17     Given the fields `dotless_text`, produce the fields `dotted_text`.
18
19
20 User message:
21
22 [[ ## dotless_text ## ]]
23 أريد أن أعلم الـرسم بلغة ناسون
24
25 Respond with the corresponding output fields, starting with the field `[[ ## dotted_text ## ]]',
26     and then ending with the marker for `[[ ## completed ## ]]'`.
27
28 Response:
29
30 [[ ## dotted_text ## ]]
31 أريد أن أتعلم البرمجة بلغة بايثون
32
33 [[ ## completed ## ]]

```

Code 2: Actual prompt structure generated by DSPy as in Code 1

```

1 mapping_desc = f"""
2     Letter mappings (undotted possible dotted forms):
3     - ب ت ث (BAA_RASM)
4     - ج ح خ (JEEM_RASM)
5     - د ذ (DAL_RASM)
6     - ر ز (RAA_RASM)
7     - س ش (SEEN_RASM)
8     - ص ض (SAAD_RASM)
9     - ط ظ (TAA_RASM)
10    - ع غ (AIN_RASM)
11    - ف (FAA_RASM)
12    - ق (QAF_RASM)
13    - ي ى ئ (YAA_RASM)
14    - و ؤ (WAW_RASM)
15    - ه ة (HAA_RASM)
16    - ن (NOON_RASM)
17    - ك (KAF_RASM)
18    - ل (LAM_RASM)
19    - م (MEEM_RASM)
20    - ء (HAMZA_RASM)
21    - ا (ALEF_RASM) آ إ ؤ ا ا ا a"".strip()
22
23 class DetailedArabicDotingSignature(dspy.Signature):
24     dotless_text = dspy.InputField(
25         desc=f"Arabic text without dots (Rasm) - simplified "
26         f"letter forms using direct character mapping as "
27         f"in the following:\n\n{mapping_desc}"
28     )
29     dotted_text = dspy.OutputField(
30         desc="Properly dotted Arabic text with correct "
31         "diacritical marks restored based on context "
32         "and meaning"
33     )

```

Code 3: Detailed prompting signature with explicit character mappings

```

1 System message:
2
3 Your input fields are:
4 1. `dotless_text` (str): Arabic text without dots (Rasm) - simplified letter forms using direct
   character mapping as in the following:
5
6 Letter mappings (undotted possible dotted forms):
7   - ب ت ث (BAA_RASM)
8   - ج ح خ (JEEM_RASM)
9   - د ذ (DAL_RASM)
10  - ر ز (RAA_RASM)
11  - س ش (SEEN_RASM)
12  - ص ض (SAAD_RASM)
13  - ط ظ (TAA_RASM)
14  - ع غ (AIN_RASM)
15  - ف (FAA_RASM)
16  - ق (QAF_RASM)
17  - ي ي ئ (YAA_RASM)
18  - و ؤ (WAW_RASM)
19  - ه ة (HAA_RASM)
20  - ن (NOON_RASM)
21  - ك (KAF_RASM)
22  - ل (LAM_RASM)
23  - م (MEEM_RASM)
24  - ء (HAMZA_RASM)
25  - ا إ آ (ALEF_RASM)
26 Your output fields are:
27 1. `dotted_text` (str): Properly dotted Arabic text with correct diacritical marks restored based
   on context and meaning
28 All interactions will be structured in the following way, with the appropriate values filled in.
29
30 [[ ## dotless_text ## ]]
31 {dotless_text}
32
33 [[ ## dotted_text ## ]]
34 {dotted_text}
35
36 [[ ## completed ## ]]
37 In adhering to this structure, your objective is:
38     Given the fields `dotless_text`, produce the fields `dotted_text`.
39
40 User message:
41
42 [[ ## dotless_text ## ]]
43 أريد أن أعلم الربح بلغة بايثون
44
45 Respond with the corresponding output fields, starting with the field `[[ ## dotted_text ## ]]',
   and then ending with the marker for `[[ ## completed ## ]]'`.
46
47 Response:
48
49 [[ ## dotted_text ## ]]
50 أريد أن أتعلم البرمجة بلغة بايثون
51
52 [[ ## completed ## ]]

```

Code 4: Actual prompt structure for detailed signature in Code 3

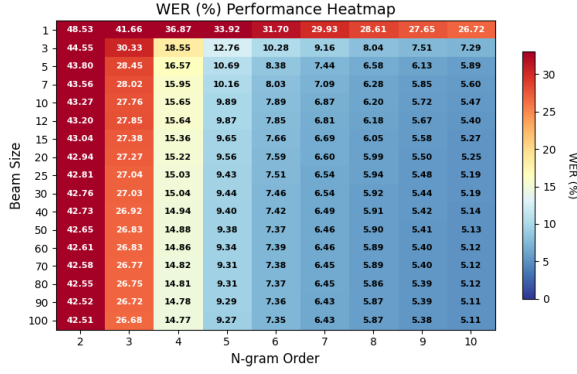


Figure 10: ngrams results on the validation dataset with various beams and ngram orders

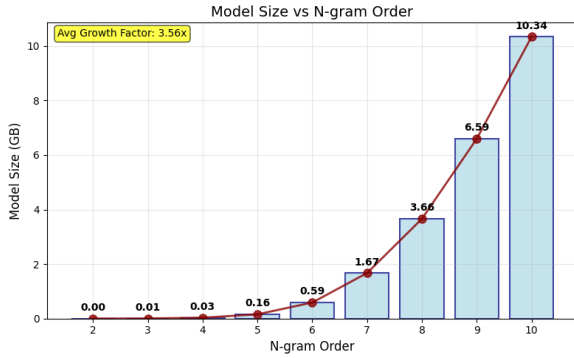


Figure 11: Model size growth as a function of n-gram order, showing an average growth factor of 3.56

D Results and Analysis on Validation Set

D.1 N-grams details

Figure 10 presents n-grams performance across varying beam sizes and orders. From this figure, WER converges beyond certain beam sizes (around 30 beam size) for each order, with orders above 8 yielding negligible returns. The most substantial improvements occur from orders 2-4 and 5-8, improving from 42% to $\approx 6\%$ WER, after which performance plateaus. However, higher orders incur substantial computational costs, as model size grows exponentially at a factor of 3.56 per order (Figure 11), reaching 1.3 GB at order 10 versus 0.01 GB at order 3. We decided to experiment with variations of beam sizes along with 3,4,6, and 8 grams on our test sets.

D.2 Sequence Labeling

For LSTMs Table 5 shows BiLSTM performance across layer depths. The most significant improvement occurs from 1 to 2 layers, with marginal gains beyond 3 layers. The last ablation with 6 layers improves by less than 0.1 WER compared to 5

layers indicating a saturation. BiLSTMs achieve fast inference (0.022-0.076 seconds) due to parallel sequence processing, contrasting with the autoregressive nature of LLMs and n-grams.

Table 5: Performance of BiLSTM models with varying layer depths on the validation dataset

Layers	WER	CER	DotER	Cost (ms/char)
1	9.69	1.87	2.22	0.0591
2	6.34	1.24	1.47	0.0856
3	6.11	1.18	1.40	0.1163
4	5.66	1.08	1.29	0.1454
5	5.43	1.03	1.22	0.1813
6	5.35	1.02	1.21	0.2093

For Transformers, Table 6 shows Transformer performance across layer depths on the validation set. The most significant improvement occurs from 3 to 6 layers, dropping WER by 1.29 points, with marginal gains beyond 9 layers. Transformers achieve fast inference (0.021–0.045 ms/char) due to parallel sequence processing, contrasting with the autoregressive nature of LLMs and n-grams. Compared to BiLSTMs, the 12-layer Transformer outperforms the best 6-layer BiLSTM on validation (4.03 vs. 5.35 WER) while remaining cheaper per character.

Table 6: Performance of Transformer models with varying layer depths on the validation dataset

Layers	WER	CER	DotER	Cost (ms/char)
3	6.01	1.15	1.36	0.0211
6	4.72	0.89	1.06	0.0268
9	4.32	0.80	0.96	0.0333
12	4.03	0.74	0.88	0.0400

D.3 Self-supervised learning via CANINE

For CANINE, Table 7 reports the two pretrained variants on validation. CANINE-S (subword-loss pretraining) slightly outperforms CANINE-C (character-loss pretraining), reducing WER from 4.85 to 4.57 with matching drops in CER and DotER; the same ordering holds on the test set, with CANINE-S at 7.48 WER vs. 7.66 for CANINE-C. The gap is small (<0.3 WER), suggesting the choice of pretraining objective has limited downstream effect for rasm-to-dotted recovery. Both variants are roughly $2\times$ more expensive per character than our 12-layer Transformer (~ 0.08 – 0.10 ms/char vs. 0.04

ms/char), yet do not improve on it—placing them between the BiLSTM and Transformer families on accuracy, but at the high end of inference cost.

Table 7: Performance of CANINE variants on the validation dataset

Variant	WER	CER	DotER	Cost (ms/char)
CANINE-C	4.85	0.92	1.10	0.0824
CANINE-S	4.57	0.86	1.02	0.0894

D.4 LLMs

For LLMs, Figure 12 presents results for 11 LLMs across different few-shot configurations and prompt styles. Top performers are the flagship LLMs (Claude Sonnet 4, Gemini 2.5 Flash Preview) where they achieved WER around 15%-21%, substantially worse than optimal n-grams (WER $\approx$ 6%). Performance varies widely (15%-75% WER), with smaller models struggling significantly. Few-shot examples beyond 3-5 yield inconsistent improvements, sometimes degrading performance (e.g., Qwen 3 235B), while prompt style variations show minimal impact.

E Comparison with the BiGRU Baseline

Setup. We take the best configuration of each non-LLM method from Table 1, namely the 8-gram with beam 60, the 6-layer BiLSTM, the 12-layer Transformer, and CANINE-s, and evaluate them without any retraining on the ATB test set of Alhathloul and Ahmad (2022), which comprises 3,538 samples and 323,134 characters. Dots are removed from their gold with the same procedure used throughout this work, and both inference and scoring reuse the pipeline applied to our own test set, so the numbers remain directly comparable with Table 1.

Contamination test. Their benchmark spans four corpora, three of which, poetry, Tashkeela, and the Quran, are drawn from sources our models train on. Our training corpus holds 3.59M documents, which makes brute-force per-sample string matching impractical, and exact-line matching would under-report the overlap in any case. We therefore measure overlap at the level of word n-grams and flag a test sample as contaminated when at least 80% of its n-grams appear in the training corpus. Both sides are normalized identically before comparing so that orthographic differences between

the two corpora do not mask genuine overlap. At $n = 8$ the flagged share reaches 99.5% for poetry, 78.8% for the Quran, and 51.9% for Tashkeela, against 0.0% for ATB. We tried multiple n-grams with $n \in \{4, 8, 13\}$ and found consistent outcomes, with ATB never exceeding 0.67%, which leaves ATB as the only fair basis for comparison.

Evaluation alignment. The gold text of their benchmark has been passed through a lossy orthographic normalization that folds hamza and madda on alef into a bare alef ($\lrcorner$, $\lrcorner$, $\lrcorner$ to $\lrcorner$) and taa-marbuta into haa ($\lrcorner$ to $\lrcorner$). Under our setup, we restore both of them, since $\lrcorner$, $\lrcorner$, and $\lrcorner$ share one *rasm* and so do $\lrcorner$ and $\lrcorner$. Thus, scoring our output against their gold without any normalization would count every such correct restoration as an error. We therefore pass both the gold and our predictions through the same two normalizations before scoring, so that only the remaining dotting decisions are counted.

Table 8: Performance on the ATB test set under the aligned evaluation setup. Their results are quoted from Table 5 of the original study.

System	WER	CER	DotER
<i>Ours</i>			
BiLSTM 6L	3.17	0.69	0.82
Transformer 12L	3.28	0.72	0.86
8-gram, beam=60	3.43	0.78	0.93
CANINE-s	5.96	1.34	1.59
<i>Theirs (Alhathloul and Ahmad, 2022)</i>			
BiGRU	9.10	2.60	5.30

Results. Table 8 reports the aligned results. All four of our methods outperform the reported BiGRU results in their study, with the 6-layer BiLSTM the strongest at 3.17 WER and 0.82 DotER against 9.10 and 5.30, respectively, and CANINE, the weakest of ours here, still ahead at 5.96 WER.

Error attribution. From the above results, we noticed that CANINE’s placement in Table 8 is inconsistent with its behaviour in Table 1, where it outperforms every n-gram configuration. To investigate, we examine where each system’s errors fall relative to the letters the alignment normalizes. Table 9 splits each system’s character errors into those falling on $\lrcorner$ and $\lrcorner$, the two letters the benchmark normalizes, and errors on other letters. Between 85% and 89% of all errors are of the first kind for the

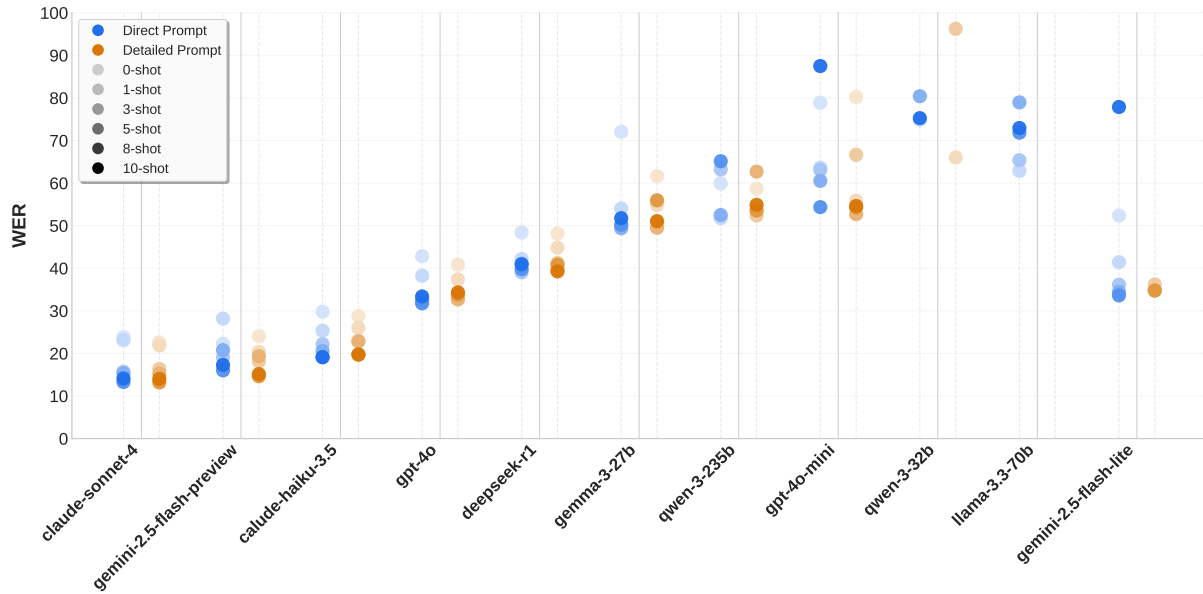


Figure 12: WER performance of various LLMs across different few-shot configurations and prompt styles on the validation dataset

n-gram and the two sequence labelers. CANINE contrasts them at 79.9%, so although it makes fewer errors overall than the n-gram, more of them fall on letters the alignment leaves untouched, which is what moves it below the n-gram in Table 8.

Table 9: Character-level error attribution on ATB. The middle column counts errors on the two letters the benchmark normalizes, and the last column counts the errors on the other errors.

System	All errors	l and o errors	Other errors
BiLSTMs	13,194	11,215 (85.0%)	1,979
Transformers	13,852	11,989 (86.5%)	1,863
n-grams	19,517	17,384 (89.1%)	2,133
CANINE-s	15,107	12,075 (79.9%)	3,032

F LLM Usage Declaration

ChatGPT and Claude were used in proof reading the manuscript and to assist constructing the code for the experiment.