UPMVM: A Metrics Verification Model for Urdu Poetry

Asia Zaman [1*], Zia-Ud-Din[1], Sajid Iqbal[2*], Asma Al Shuhail[2]

[1] Department of Computer & Information Technology, Faculty of Computing, Gomal University, Dera Ismail Khan, KPK (Pakistan)
[2] Department of Information Systems, College of Computer Science & Information Technology, King Faisal University, Al-Ahsa 31982 (Saudi Arabia)

* Corresponding author: aasiazaman123@gmail.com(A.Zaman), siqbal@kfu.edu.sa (S.Iqbal)

Received 14 November 2024 | Accepted 23 March 2025 | Published 15 September 2025

Abstract

Urdu poetry retains a prominent position in the cultural heritage of Urdu language. Rhyme schemes and meters are frequently employed in poetry, which follow specific patterns and structures. Natural Language Processing has the capacity to recognize and analyze these patterns, which is beneficial in the investigation of poetic forms. This research presents the UPMVM (Urdu Poetry Metrics Verification Model), a novel rule-based architecture, designed for detecting meter of any given Urdu ghazal verse. In this work, we propose an algorithm that consists of sixteen steps that identifies the Arud meter in the Urdu verses using a custom developed system. This application will not only assist professional poets but also enable students to examine poetry within the framework of prosody principles. The accurate analysis of the prosody of any poetry relies on the act of uttering words rather than on a written record. UPMVM consists of two phases: 1) The primary objective of the initial phase is to consolidate all available literature of the Arud system into a unified digital platform, then develop individual and combined DFA of each identified meter for pattern recognition; 2) the second phase is about the algorithmic implementation. All these rhythmical patterns are matched with 290 Arud meters and their sub-meters developed during this study. The implementation strategy of phase 2 comprises of five essential sub-phases including tokenization, orthography, syllable identification, weight assignment, and meter detection. For evaluation of the proposed method, three different datasets are used for feature extraction, token identification and performance measurement for identification of rhythmic patterns in Urdu poetry. The UPMVM model reached to promising outcome with an average accuracy of 94%.

Keywords

Arud Meters, Natural Language Processing, Pattern Matching, Poetry, Urdu Ghazal.

DOI: 10.9781/ijimai.2025.09.001

I. Introduction

Urdu poetry is based on Arud system derived from Arabic and Persian languages. Arud System is the rhythmic pattern found in any poetry. The knowledge of Arud is crucial in Urdu poetry because it provides an appropriate process for ensuring that the poet's work complies with poetic weight regulations [1]. Poetry consists of tone, metrical forms, melody, imagery, and symbolic representation [2]. Al-Khalil bin Ahmed is the founder of Illm-e- Arrud [3]. He designed this knowledge by using the ideologies of the tune. He used basic knowledge of Arabic poetry; his focus was that every verse ensures some patterns to give proper weights. Poetic Weight (PW) is a poetic meter which is the rhythmical arrangement of (متحرک) mutaharik and (ساکن) sakin letters into verse parts lines [4]. Later on, these rules, with some modification and addition, got popular in Persian poetry [5]. The origin of Urdu poem is believed to be couplet poems (a form of Arabic poetry). Persian language influenced the Turkish and Indian region and this art of versification got popular in these languages as well [6]. Urdu poetry deals with Arud meter. Arud is the study of poetic meter that is practiced in Arabic, Turkish, Persian, Urdu, Hindi, Punjabi, and other languages of South Asia. Arud meter is a standard pattern of sounds established by poetry experts [7]. These meters are composed of feet [6]. The feet are the basic sound patterns composed of short or long syllables. Short syllable is represented by "1" and long syllable with "2". If a meter is developed by repetition of a single foot, like فعو لن فعو لن فعو لن فعو لن (122 122 122 122), it is called a simple meter. On the other hand, if a meter is formed by combination of different feet, it is called a compound meter, such as فاعلاتن مفاعلن فعلن (2122 1212 22). Different poetry styles use different number of feet. The line of 8 feet is called Musaman (مثمن), while line of 6 feet is referred as Musads (مسدس). Meter nomenclature is based on number of feet. There are 19 basic Arud meters [4] whereas one more meter Jameel is added recently. These meters were designed to make them easy to understand without referring context. The feet join together in different orders and quantities to constitute the meters. If the poetry were restricted to 19 basic meters only, it had limited choice for poets. All meters consist of multiple sub-meters. These sub-meters show different arrangements of feet with its zehaaf (زحاف) or catalexis.

Zehaaf word means amendment from the original one. Catalexis is the name given to any irregular change occurring in the meter. There are specific rules applied on the zehaf [8]. Most commonly, following three rules or their combinations are implemented to make a zehaf.

1. Making mutaharik words sakin.
2. Eliminating sakin or mutaharik letters.
3. Adding more letters in feet/affails.

The use of catalexis or "zehaaf" generates multiple of new meters and an acceptable range of meters exceeds 100 [3] and using all possible zehaaf of basic meters we gather around 290 meters and sub-meters from basic feet. The complicated nature of these regulations emerges from the fact that the primary criterion for justifying the phonological stability of verses is the accumulated effect of sounds as a whole, rather than the specific words or letters used. To determine and quantify the weight and rhythm of sounds, the poem is analyzed by dividing it into words (Tokens), and further breaking down the words into syllables. A syllable is a unit of sound in prosody that can be classified as either a long syllable, a short syllable, or a flexible syllable. The representation of weight of syllables is given in Table I. A long syllable is formed when two consonants are combined with a short vowel. A syllable that consists of only one vowel or consonant is referred to as a short syllable. A flexible syllable consists of a consonant and a long vowel. Its pronunciation can vary, being either long or short depending on how it is spoken. After retrieving token syllabification, we assign weights to each identified syllable according to Table I to get the numeric pattern of the meter for further meter detection.

TABLE I. Syllable Representation

Syllable	Weight Representation
Long	2
Short	1
Flexible	*X*

Our approach firstly identifies verse tokens by splitting functions, then applies Arud rules on tokens to get token-taqti that contains only those words which are phonetically effective. Token-taqti describes the verse syllables. The identified syllables are subsequently measured in quantity in order to establish the verse pattern. Subsequently, in order to facilitate verse pattern matching, the state sequence of the Deterministic Finite Automaton (DFA) associated with each of the 290 identified meters is retained and subsequently retrieved.

The main contribution of the model involves organizing and preserving Urdu meters and their variations, creating a unique pattern for each meter. Using verse tokens model a comprehensive dataset for an Urdu lexicon database is developed. Analytical approaches are used to verify the authenticity of Urdu verse meters, and the use of algorithms in Urdu Poetry Metrics Verification (UPMVM) systems enhances metric management and analysis. One main influence of UMPVM is improving educational tools and information retrieval mechanisms. The rest of this article comprises 6 sections. Section II contains a brief overview of related work. Section III illustrates the datasets used in the proposed methodology. Next section IV describes the proposed methodology which includes the steps and algorithms used in the UPMVM implementation. Section V further elaborates on the methodology step by step by running a sample example test-case. Section VI shows the evaluation of test cases results using different evaluation measuring techniques. Finally, Section VII presents the conclusion and future work.

II. Related Work

Currently, there is a lack of research explicitly focused on analyzing the prosody of Urdu poetry using computational methods. In Urdu language hot area of research is stemming [9][10], POS tagging [11], parsing [12] and context analysis. However, there are research papers dealing with content analysis, authorship attribution of Urdu poetry using machine learning algorithms. In 2020 Nida Tariq's [12] and in 2019 Momna Dar's [1] studies use a variety of machine-learning approaches to determine the poet's name. To categorize the poet's name, they employ many classification techniques, including Decision Tree, Nave Bayes, SVM, Random Forest, and KNN. In another work [13] Agha Ali Raza applied N-gram-based algorithms to corpora for applied authorship attribution. Shahid Rabbani [14] in 2006 explored Urdu poetry data to provide numerical insight into ghazal features analyzed using a multidimensional scale to identify the linguistic diversity.

The Arud system has its origins in the Arab culture. Hence, we have identified the scholarly research primarily focused on the Arud meter in Arabic [15]-[25] as well as its use in Punjabi [26][27] and Ottoman languages [28]. Additionally, software systems have been created to analyze the prosody of Arabic poetry. A computing system was built at Yarmouk University in 2003. Unfortunately, the test data for that system are no longer accessible [29]. An expert system named ALAroud was developed in 2009, which specifically met the requirements of Al-Khalil ibn Ahmad al-Farahidi, including the use of diacritic markings. In 2010, a system called EHST was developed [30]. This approach not only identified the meter, but also emphasized the incorrect placement of words inside the poem. In 2013, a method called "Finding Arabic Poem Meter using Context-Free Grammar" [15] was created using regular expressions (REs) and Context-Free Grammars (CFGs) where pipeline-based text processing technique was employed. In this work, the initial stage was the conversion of the stanza into a metrical pattern. Subsequently, the verse was segmented, and ultimately, the meter was ascertained. The system achieved an accuracy rate of 75% for the given input test data. Subsequent advancements, exemplified by the system presented in 2014, prioritized accuracy and efficiency in Arabic poetry prosody analysis. Dahab et al. [16] and Belal [30] contributed to this progress by proposing algorithms utilizing finite state automata and focusing on the first part (Sadr) of Arabic poetry verses. In recent studies [23] [24] for Arabic meter detection techniques deep learning models are used and generate results with high accuracy. Recently in 2024 [23] Mutawa uses classical Arabic poetry that follows one of the 16 meters, with metering being the rhythmic framework. A deep learning model was implemented using TensorFlow on a large dataset of Arabic poetry. Character level encoding was used for full-verse and half-verse classification, with the Bi-LSTM model showing the best accuracy, with 97.53% for full-verse and 95.23% for half-verse. In 2019 [24] Waleed introduces machine learning models RNN in detecting poems meters from plain text. This is a step forward for machine understanding and synthesis of languages in general, and Arabic language in particular. Machine learning networks successfully classified 16 Arabic and 4 English poem meters with 96.38% and 82.31% accuracy, respectively, using massive datasets of over 1.5 million verses from nontechnical sources and unstructured formats. Zeyada et al. [25] introduce a new AI system called "IMAP" to automatically recognize the Arud of modern Arabic poems, a challenge in this field. The system uses machine learning techniques to identify Arud, a group of syllables that form a prosodic unit, with an accuracy of 99%. Another work [31] identifies the poetic meter in spoken Arabic poetry, this research suggests a voice-based methodology. Three meters spoken aloud by two speakers are among the 230 samples drawn from ten poems that the model employs. The model employs a support vector machine (SVM) classifier, a long short-term memory classifier, and linear prediction cestrum coefficient and Mel frequency cepstral coefficient features. The speaker-dependent strategy achieves the highest accuracy in the SVM model and surpasses the performance of state-of-the-art research by 3%. An analogous methodology is recognized in the field of Turkish literary academia. In 2012, Kurat [28] introduced a system for identifying the Arud meter in Diwan poetry, a crucial aspect of the old Turkish literary heritage. The initial input consists of verses written in Ottoman character, which are subsequently transformed into Latin Transcription Alphabets (LTA) through transliteration. Following transliteration, the text is subjected to metrical analysis in order to determine the meter and detect any associated deviations or inconsistencies. In 2020, Muhammad Raihan [26] created a web application specifically designed to detect Arud meters in Punjabi Ghazal poetry. The approach entails applying orthographic changes, scanning verses, and utilizing nested iterations to synchronize rhythmic patterns with 37 Arud meters. The automated methodology surpasses human alternatives in terms of efficiency. A research based on observation and experience discovered that 83% of the dataset used for testing could be accounted for by 5 particular Arud meters. In 2023, Ayush [23] employed machine learning techniques to automate the categorization of Punjabi ghazels. The dataset was divided into four distinct categories. In Latent Dirichlet Allocation scenarios, the Support Vector Classifier (SVC) consistently achieved a higher accuracy rate of 82.17% compared to TF-IDF. Despite no scientific research on detecting Urdu poetry metrics, websites like Aruuz [32] and Rakhta [33] help poets determine their work's meter. Aruuz offers tools for detecting overlap weight, closest overlap, word-matching corrections, and single word overlap. Aruuz dataset [34] contain 80k Urdu words. On the other hand, Rakhta provides a comprehensive online resource for Urdu literature, including translations, dictionaries, audio and video resources. Both sites aim to promote and uphold Urdu's cultural heritage where Rakhta has recently launched, in April 2023, the beta version of its project Taqti, to detect the meter of any given verse. The fundamental knowledge of the Arud meter was acquired from chapters 1-7 of the book "Pritchett and Khaliq" [6] and chapters 2-5 of Captain Pybus' book [7].

In [35] one of the recent paper Maged used RRN and BiRRN for tasks such as text classification, named entity recognition, or language modeling, which might be adaptable for meter classification to detect Arabic poetry meter. Urdu Arud meters classification work done in different studies [36] [37] to explore the problems faced in Arud system [38].

III. Evaluation Dataset

The UPMVM model utilizes three datasets named UD1, UD2 and UD3. UD1 is primarily used to collect and retrieve 290 poetry meters (rhythmic patterns [بحر) and their corresponding feet. Existing literature and expert's feedback used to generate UD1. Other uses of this dataset include the design of DFA state function sequences with terminal state information to align the identified verse meters. UD2 is collected from Zeeshan's repository [34] and updated. This update process involves the parsing and tokenization of UD2 dataset. Arud rules are then applied to each token for its discretization, resulting into a tok-taqti which is then added to vocabulary if it is not already present. This dataset UD2 contains 80,000 Urdu words along with their diacritics, language information, and word variations. The third dataset UD3 is used to evaluate the proposed model. UD3 contains 500 verses including structure-based genre, collected from published books [3][4][5][6]. The dataset contains verses that have undergone rigorous reviews by poetry specialists, and each verse is annotated with poetic metrics like syllable count, rhyme scheme, meter pattern, and metadata about genre, style, and period. UD3 serves as a starting point for validating the UPMVM. Table II, Table IV and Table VI present the important features of all three datasets. While Table III, Table V and Table VII explore the record samples of all datasets.

Our understanding of the metrical structure of verses is not restricted to particular poets or periods in poetry; rather, it applies to every Urdu Ghazal verse. The dataset utilized for model evaluation comprises works of early Urdu poets which are done during the nineteenth century, with particular attention on its structural components.

TABLE II. UD1 Description

Feature	Description	Measure
Dataset:	**UD-1**	
No. of meters	Identified Urdu meters and its Sub meters	290
Meter Language	Urdu, English	2
Size of dataset		98KB
DFA terminal state	Final state showing the acceptance of pattern	
DFA states pattern	Individual meter state functions pattern stored	Example state sequence [0 33 34 35 36 37 38 39 601 602 603…]

TABLE III. UD1 Record Sample

Meter ID	Meter EngName	Meter UrduName	DFA states Sequences	Terminal	Meter feet /Afail افاعیل
101	Hazeej Musamn Akhrab	ہزج مثمن اخرب	[0 33 34 35 370 372 374 384 385 386 387 379 380 381 382]	382	مفعول مفاعیلن مفعول مفاعیلن
66	Kamil Musamn Muzmar Salim Alakhir	کامل مثمن مضمر سالم الاخر	[0 1 201 202 233 234 270 271 272 273 274 275 276 277 278 279 280 281 282]	282	متفاعلن مستفعلن متفاعلن مستفعلن
35	Rajez Musaman Matawa Mazal	رجز مثمن مطوی مذال	[0 33 34 55 56 57 58 112 113 114 115 116 117 598]	598	مفتعلن مفتعلن مفتعلن مفتعلان
9	Mutakarib Musaman Maqsor	متقارب مثمن مقصور	[0 1 2 3 4 5 6 7 8 9 10 11 31]	31	فعولن فعولن فعولن فعول
56	Ramal Musaman Makhboun Mahzof Maskin	رمل مثمن مخبون محذوف مسکن	[0 33 59 177 178 218 219 220 221 648 649]	649	فاعلاتن فعلاتن فعلن

TABLE IV. UD2 Description

Dataset:	UD-2	
Feature	**Description**	**Measure**
No. of token	Urdu token with diacritics	80k tokens
Token Language	Urdu, Persian, Arabic, Sanskrit, English	Languages merged
in Urdu		98KB
Size of Dataset		8.50 MB
Tok-Taqti	Basic Arud rules applied on token to get tok_taqti	Example state sequence [0 33 34 35 36 37 38 39 601 602 603...]

TABLE V. UD2 Sample Record

Token ID	Token	Muarrab	Taqti	Language
45008	مراعات	مُراعات	مُرا+عات	عربی
7	آؤ بھاؤ	آؤبھاؤ	آ+اوبھا+ او	سنسکرت
23	آئندہ	آئِندَہ	آ+اِن+ دَہ	فارسی
124	آب دیدہ	آب دِیدَہ	آب + دی + دَہ	فارسی
178	آبائی	آبائی	آ+با+ای	عربی
333	آتھر	آتھَر	آ+تھَر	انگریزی
364	آخرشب	آخِرِشَب	آ+خِرے+شَب	اردو
441	آدمیت	آدْمِیّت	آد+می+یَت	عبرانی
476	آراضی	آراضی	آ+را+ضی	عربی
1075	آڑو	آڑُو	آ + ڑُو	ہندی
64500	وعظ	وَعظ	وَعظ	عربی

TABLE VI. UD3 Description

Feature	Description	Measure
No of Verses	Number of verses used for result analysis	500
Meter Language	Urdu	
Size of Dataset		5 KB
Verse Annotation	Randomly selected Ghazal verses	
No. of meters	Most frequently used meters in Urdu	9

TABLE VII. UD3 Sample Record

V #	Verse	Verse Meter	Verse Afail	Verse Syllable quantification
1	خودی کو کر بلند اتنا کہ ہر تقدیر سے پہلے	ہزج مثمن سالم	مفاعیلن مفاعیلن مفاعیلن مفاعیلن	1 2 2 2 1 2 2 2 1 2 2 2 1 2 2 2
2	کھول آغوش نہ تو مجھ سے رکاوٹ سے لپٹ	رمل مثمن سالم مخبون محذوف	فاعلاتن فعلاتن فعلاتن فعلن	2 1 2 2 1 1 2 2 1 1 2 2 2 2
3	ستم کو ہم کرم سمجھے جفا کو ہم وفا	ہزج مثمن مقبوض محذوف	مفاعلن مفاعلن مفاعلن فعل	1 2 1 2 1 2 1 2 1 2 1 2 2 1
4	صبح نسیم کی بہار ساتھ لے آئی بوئے یار	رجز مثمن مطوی مخبون	مفتعلن مفاعلان مفتعلن مفاعلان	2 2 2 1 2 1 2 1 2 2 2 1 2
5	دل عبادت سے چرانا اور جنت کی طلب	رمل مثمن محذوف سالم الاخر	فاعلاتن فاعلاتن فاعلاتن فاعلن	2 1 2 2 2 1 2 2 2 1 2 2 2 1 2

During the development of our dataset, we have observed that current state of Urdu poetry resources is inadequate for the purpose of meter detection and Urdu related NLP. In this context, our dataset, for meter detection, is an initial data collection.

IV. Methodology

Identifying the Arud meter in poetry requires a sophisticated technique of scansion. The fundamental concept underlying this approach involves transforming the verse into a corresponding rhythm pattern. The primary contribution of this work is the development of an innovative method known as UPMVM, which seeks to identify every original or variation meter found in any verse in conventional Urdu poetry. UPMVM consists of two primary modules: the first one is used for gathering Urdu meter data, while the second one is used to implement algorithm for developing applications to analyze meter detection. The rule-based approach used in our method encapsulates the domain-specific knowledge or heuristics, guiding the system's behavior in processing and analyzing data. Rule-based systems are characterized by their explicit representation of knowledge in the form of if-then rules or logical statements.

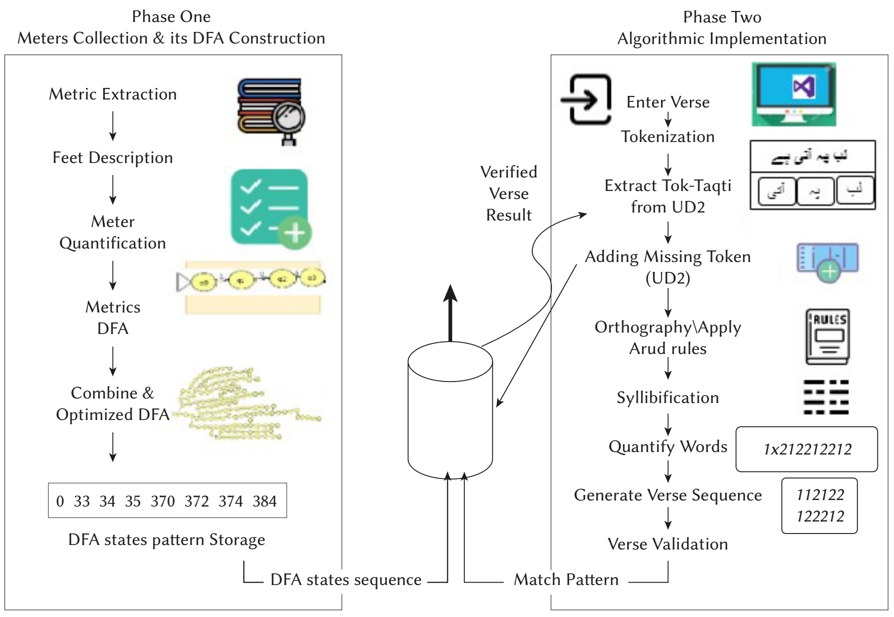


Fig. 1. UPMVM Steps.

The literature highlights the need for further research on the prosody of Urdu poetry due to limited data resources and the absence of a standard model for verifying written Urdu poetry verses. UPMVM comprises of two main phases following main steps as shown in Fig. 1. First six steps comprise phase I while remaining 10 steps make the phase II.

The first phase involves gathering metrics related to Urdu poetry and constructing its DFA, while the second phase involves implementing algorithms to create an application that can identify verse meters. Each phase has its own unique set of steps. The initial phase of meter collection and DFA construction goal is to compile metrics and their sub-meters for Urdu poetry from published works, quantify them and then each detected metric's DFA is created. The second phase of algorithmic implementation is to develop a tool by executing algorithms for validating Urdu verse meter. Both phases contain the number of steps shown in Fig. 1, which describes the pictorial representation of UPMVM.

A. Phase One

Phase I of the UPMVM comprises six steps aimed at meter collection and constructing meters DFA for pattern recognition.

Step1: *Collection and Classification of Urdu Meter and its Sub-meters*

This initial phase entails the comprehensive gathering of data pertaining to Arud meter, a fundamental aspect of Urdu poetry. The compilation process involves sourcing information from literary works and consultations with experts [6][7][8][21][3]. A thorough examination of available materials has resulted in the identification of 290 meters. Each associated with its respective units of measurement, namely feet or Afails. The classification of meters is based on the foundational principles of feet and zeehaf. Basic meters such as لماک kamil, جزه Hazej, and براقتم Mutakarib are introduced, alongside the corresponding number of feet, such as Musadas for six feet and Musaman for eight feet. Additionally, meters nomenclature is established, incorporating the alignment of zeehaf points within meters for precise identification and categorization. A detailed presentation of meter nomenclature and quantification is provided in Table VIII.

Step 2: *Description and Analysis of Meter Contents (Afail – افاعیل)*

This step involves the meticulous examination of the constituents of each identified meter and its sub-meters. The composition of

TABLE VIII. Meter Nomenclature & Quantification

1	Mutakarib Musadas متقارب مسدس	فعولن فعولن فعولن 1 2 2 1 2 2 1 2 2 (Left to right)
2	Mutakarib Musaman Aslam Maqsor متقارب مثمن اثلم مقصور	فعلن، فعولن، فعولن، فعول 2 2 1 2 2 1 2 2 1 2 1

feet is analyzed with regard to their short and long syllables. For instance, the meter pattern فعولن is dissected into its constituent syllables (ف،عو،لن), with a quantification of 1 2 2, denoting the sequence of short, long, and long syllables.

Step 3: *Quantification of Feet in Meters*

Once the feet have been accessed, the following step is to quantify them in meters. The quantification of each meter is represented as a sequence of 1s and 2s. As discussed above, a short syllable is represented by the number 1, whereas a long syllable is represented by the number 2. There will be no alteration in feet, so there will be no flexible syllable in Afail/feet composition. For example, if we have a metric quantification string pattern as in Fig. 2: 1 2 1 2 2 1 2 1 2 2 feet/Afails فعول فعلن فعول فعلن we parse this متقارب مقبوض اثلم meter string into tokens in the form of afail and then quantify them. The JFLAP [39] software was used to build individual as well as combined DFA for all 290 meters listed in dataset UD1.

Step 4: *Construct a Deterministic Finite Automaton (DFA)*

A DFA is constructed for every meter that has been identified in the preceding stages. Each DFA is used for pattern recognition of the meters through its state sequences. Fig. 2 shows individual meter (متقارب مقبوض اثلم) Mutakarib Maqboz Aslam's DFA.

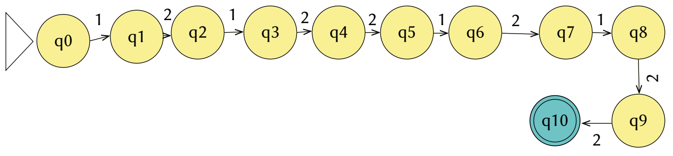


Fig. 2. Individual Meter DFA.

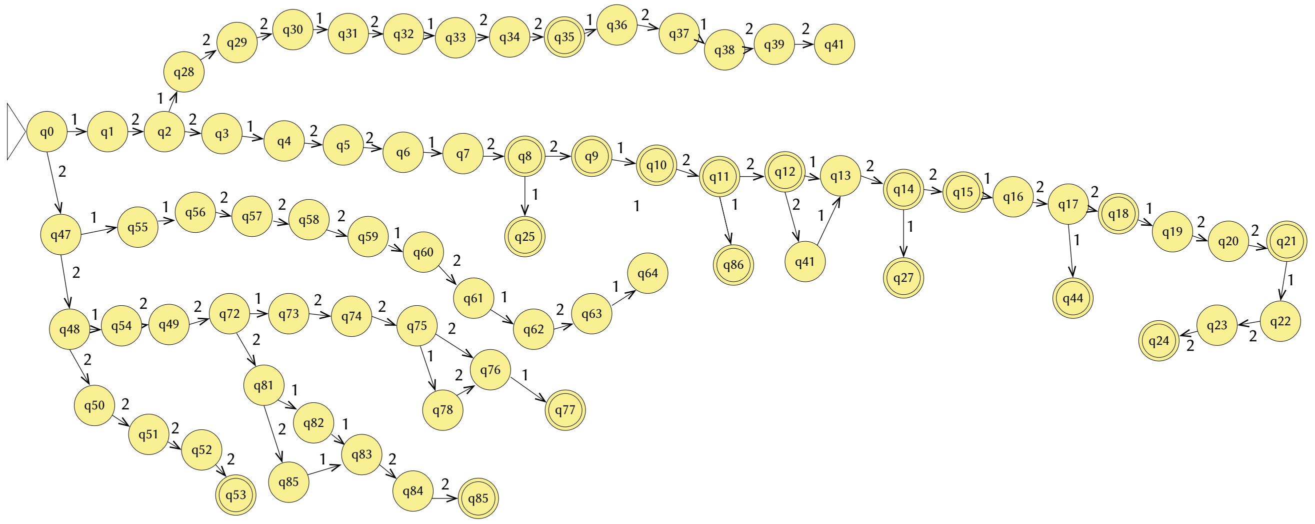


Fig. 3. Combined DFA of meter Mutakarib.

Step 5: *Integrate the individual DFA*

The individual DFAs of all meters and sub-meters are integrated to formulate an optimal combined pattern. Firstly, combined DFA of all basic 20 meters with their sub-meters is created. Lastly overall combined patterns are created for showcases of the interconnectedness of meters and their corresponding sub-meters. One of the basic Meter متقارب Mutakarib ‘s Combined DFA is depicted in Fig. 3.

Step 6: *Development of Database and Verification Process*

The database stores meter name with its feet/Afail and the series of DFA state patterns in combined DFA with their terminal state. The stored metric sequence will be compared with the generated verse sequence to determine whether the entered verse is written with the appropriate emphasis to match a specific meter.

B. Phase Two

This phase II of UPMVM comprises 11 steps, during which a variety of procedures are utilized for implementation. This application is designed to determine the meter of any Urdu poetry verse entered by the user. In algorithmic representation of certain terms, we use some symbols presented in form of the following equations:

- Verse $\longrightarrow vr = \sum_{i=1}^{n} tok$ (1)

  # Verse is collection of tokens

- Tokens $\longrightarrow tok = \sum_{i=1}^{n} toktaqti$ (2)

  # Token is collection of token-taqti words

- TokenTaqti $\longrightarrow toktaqti = \sum_{i=1}^{n} le$ (3)

  # Tokentaqti is a collection of letters

- Letters $\longrightarrow le = \sum_{i=1}^{n} vo, cn$ (4)

  # Letters is collection of vowel & consonant

- Vowel $\longrightarrow vo =$ {ہ، آ، ا، ی، و، ے} (5)

  # Vowel are set of certain Urdu alphabets

- Letters $\longrightarrow cn =$ {ب، پ، ت، ـ، ـ، ـ} (6)

  # All remaining Urdu alphabets which are not vowels

Step 1: *System Input (poetic verses)*

Input a Verse (*vr*) into the tool without diacritic (حرکات). *vr* is collection of verse tokens according to eq (1).

Step 2: *Tokenization*

Split verse (*vr*) into tokens (*tok*) as shown in eq (2) using ‘space’ based split function. Algorithm 1 generates number of tokens by applying space split function. Identified tokens are then matched with the dataset consisting of more than 80K Urdu lexicon [34]. If a token *tok* is found in the database, then token with diacritics is retrieved from UD2 dataset in UPMV application for further actions, otherwise step 3 invokes.

Algorithm 1: Splitting Urdu Verse

```
Data: Array of verse splitted via split function
Result: Split Urdu verse with taqti
Split() ;
String[] Patterns = vr.Split(“ ”);
foreach String tok in Patterns do
    Match tokens in the Lexicon database;
    if match found then
        extract toktaqti with diacritics;
    else
        insert missing token taqti with diacritics;
    end
end
end
```

Step 3: *Adding missing token to dictionary*

Add a missing token if the token is not found in the database. A function *Inserttaqti*( ) is called in Algorithm 2 to insert the missing token with its taqti into UD2 dataset.

Algorithm 2: Database Insertion Function

```
tok search missing token;
enter token taqti \# database with diacritics;
update database;
return missing token;
```

Step 4: *Systematic analysis of tokens*

The system utilizes a series of scansion Arud rules functions to systematically analyze all *toktaqti* words, identifying those that produce audible sounds upon pronunciation. Various scansion algorithms given below (algorithms 3-12), i.e. nasalization, implementing diacritics such as Tashdid (ّ), Kahrii Zer (ِ), Dagger Alif (ا), wow (و) construction, Kasr-e-Izafat, Silent Letters (و, ء), Alif Madd Grafting (آ), and Alif Grafting (ا), can be applied as necessary on any token *tok* during this step.

```
Algorithm 3: Nasalization Procedure
  Data: token le contain ھ
  Result: toktaqti without ھ
  if toktaqti contains ھ (Unicode $U + 06BE$) then
    remove ھ
  end
  if toktaqti contains ں (Unicode $U + 06BA$) then
    remove ں
  end
```

```
Algorithm 4: Procedure removeTashdid
  Data: toktaqti le contains tashdid
  Detected letter with Tashdid Result: Modified le without tashdid
  if le contains $U + 0651$ then
    repeat
      Repeat the le twice with diacritic $U + 0651$;
    until le twice;
  end
```

```
Algorithm 5: Procedure deal KahriiZer
  Data: toktaqti
  Result: Modified verse with Kharii Zer replaced by ی
  if verse contains "Kharii Zer" then
    Replace "Kharii Zer" with ی;
  end
```

```
Algorithm 6: Procedure DaggerAlif
  Data: toktaqti
  Result: Modified verse with Dagger Alif replaced by ا ;
  if verse contains "Dagger Alif" then
    Replace "Dagger Alif" with ا;
  end
```

```
Algorithm 7: Procedure wowConstruction
  Data: tok, le
  Result: Modified word with appended at the end if it is followed
  by le
  if tok is followed by le و then
    Append at the end of the word;
  end
```

```
Algorithm 8: Procedure KasreIzafat
  Data: toktaqti
  Result: Modified verse with KasreIzafat replaced by ے
  if verse contains KasreIzafat then
    Replace KasreIzafat with ے;
  end
```

```
Algorithm 9: Procedure silentWow
  Data: toktaqti
  Result: Modified toktaqti with removed if it is immediately after
          the letter خ
  if toktaqti contains خ then
    Remove و;
  end
```

```
Algorithm 10: Procedure silentHamza
  Data: toktakti
  Result: Modified toktaqti with removed if it is at the end of a
          word
  if toktaqti contains ء at the end of a word then
    Remove ء;
  end
```

```
Algorithm 11: Procedure AlifMaddGrafting
  Data: input toktaqti
  Result: Modified vr with vo آ replaced with vo ا under certain
          conditions
  if vr contains a toktaqti starting with vo آ then
    if the last le of previous toktaqti is con then
      Replace vo آ with vo ا in the toktaqti;
    end
  end
```

```
Algorithm 12: Procedure ReplacingHamza
  Data: toktaqti
  Result: Modified verse with Hamza ء replaced by Alif if it is not
          at the end of a token
  if a tok in the verse contains Hamza ء which is not at the end then
    Replace Hamza ء with Alif ا;
  end
```

For example in Algorithm 3 of Nasalization, Unicode U+06BE is detecting ھ while U+06BA is used for ں . According to Arud rules ھ, ں does not make any sound and omitted from the token *tok*. A Nasalization algorithms is invoked for every diacritic in the token *tok* and triggered to execute appropriate procedure when specific diacritics were applied to the tokens. When necessary, these Arud rules were implemented to *toktaqti* via an *if* − *then* condition.

Step 5: *Token scansion*

This step extracts the entirety of token scansion (toktaqti-) values from the step 4. After applying the Arud rule to the retrieved toktaqti, any of the algorithms listed above may be utilized depending on the diacritics present in the token. For toktaqti, a number of Arud rules are implemented to the *tok*. The classification of taqti words into short, flexible, and long syllables is determined by the length of the words and the position of the vowels in these toktaqti words as shown in Algorithm 13.

```
Algorithm 13: Scantokenlength Procedure
  if token.length = 1 then
    Call scanOnelength();
  end
  else if token.length = 2 then
    Call scanTwolength();
  end
  else if token.length = 3 then
    Call scanThreelength();
  end
  else if token.length = 4 then
    Call scanFourlength();
  end
  else if token.length = 5 then
    Call scanFivelength();
  end
```

The *Scantokenlength* function is responsible for determining the length of each token and then calls the corresponding functions in accordance with the length of the token. Every *Scanlength* function (scanOnelength, scanTwolength, scanThreelength, scanFourlength and scanFivelength) in Algorithm 13 is specifically engineered to evaluate the placement of consonants and vowels within the token and apply Arud rules mentioned in algorithms 3-12. The identification of *toktaqti* syllables within the token is facilitated by this analysis.

Step 6: *Taqti word differentiation*

The system differentiates *taqti* words into short, flexible, and long syllables, which are based on the position of the vowels and the length of the words. Algorithm 14 used for this procedure.

```
Algorithm 14: Syllables Identification Procedure
  all toktaqti in the vr are traversed if (con is followed by vo) then
    Mark the syllable as Long;
  end
  if (vo not at the end of toktaqti) then
    Mark the syllable as Long;
  end
  if (con alone in toktaqti) then
    Mark the syllable as Short;
  end
  if (vo at the end of toktaqti) then
    Mark the syllable as Flexible;
  end
  if (con is preceded by con) then
    Mark the Sakin char as a short syllable;
  end
```

Step 7: *Taqti- token quantification*

In this step taqti tokens are quantified according to the identified syllable by calling function *assigncode*( ). Words are differentiated as short, long, and flexible syllables as per arud rules and quantified as 1, 2, and, x respectively as shown in Algorithm 15.

```
Algorithm 15: AssignCode Procedure
  all toktaqti in the vr are traversed if (syllable is Long) then
    Set weight of syllable to "2";
  end
  if (syllable is single Short) then
    Set weight of syllable to "1";
  end
  if (syllable is Flexible) then
    Set weight of syllable to "x" // flexible weight;
  end
```

Step 8: *Sequence generation*

This step generates two sequences for flexible syllable words by calling the function *generatesequence*( ). The flexible syllable is treated as '1' in the first pattern row and then '2' in the other pattern. The same is repeated for each flexible syllable found in the verse pattern.

In summary, step 8, generates all possible sequences denoted by 1 and 2, representing syllable types. The number of generated patterns depends on the presence of 'x' flexible syllables. For example, in Verse test-case 1 of Fig. 5, a sequence like [2 2 x 2 x 2 1 1 2 2 1 2 1 2 1] generates four distinct patterns by applying Algorithm 16. These patterns are essential for understanding the rhythmic structure and meter of Urdu poetry. The generated sequence produces four possible patterns as

$$\begin{pmatrix} 2&2&1&2&1&1&2&2&1&2&1&2&1 \\ 2&2&1&2&2&1&2&2&1&2&1&2&1 \\ 2&2&2&2&1&1&2&2&1&2&1&2&1 \\ 2&2&2&2&2&1&2&2&1&2&1&2&1 \end{pmatrix}$$

```
Algorithm 16: Generate All Possible Sequences
  Data: Original Urdu verse
  Result: Verse All possible sequences
  GenerateSequence() ;
  if Vr contains X;
  then Check the number of X in verse length;
  foreach X in Vr sequence do
    interpret X with 1 ;
    interpret X with 2;
    Display all possible sequences;
```

Step 9: *Matching verse pattern*

This step compares the verse pattern generated in previous step sequentially with the series of stored DFA (step 6, UD1 dataset column # 4) by calling a *matchSquence*( ) function (see Algorithm 17).

```
Algorithm 17: DFA state $q_{0}$ function for MatchSequence
  Input: Verse DFA states matched sequence
  if next == null then
    display "No meter is matched";
    if next ==1 then
      fq1();
    end
    else
      fq(33);
    end
  end
```

Step10: *Matching verse pattern with Afail*

The system detects the exact match of the pattern with the DFA state sequence and verifies the meter name with its Afail. In this step, we can detect the meter name with its Afail of any entered verse. Overall cumulative algorithm for detecting verse meter is depicted in Algorithm 18.

```
Algorithm 18: Collective Urdu Poetic Verse Matching Algorithm
  Input: Enter Urdu verse without diacritics
  Output: Urdu Verse Matched Meter, Meter Name, Meter Feets
  Entered Urdu poetic verse;
  vr splits into tokens by split() // Apply Algorithm 1;
  for each token in the Database do
      if token found then
          Scantokenlength() // Apply Algorithm 13;
      else
          dbinsertion() // Apply Algorithm ';
      end
  end
  for each le in taqti-token do
      apply Arud rules // (algorithms 3-12, Arud rules algo applied
      where needed);
  end
  for each taqti-token in tokens do
      callassigncode() // Apply Algorithm 15;
  end
  for each sequence in original sequence do
    generatesequence(Patterns) // generate all possible verse pattern;
      for each Pattern in Patterns do
          for each state function do
              matchedSequence() // Apply Algorithm 16;
          end
          display detected meter;
      end
  end
```

Next, the Algorithm 18 is written in pseudo code form for clarity to increase readability.

```
Algorithm: Collective Urdu Poetic Verse Matching Algorithm
Input:
  Urdu verse without diacritics
Output:
  Matched Meter, Meter Name, Meter feet
Procedure:
1. Split the Urdu poetic verse into tokens.
2. For each token in the dataset:
  a. If the token is found:
    i. Apply Algorithm 13 (Scan token length).
  b. Else:
    i. Apply Algorithm 2 (missing token insertion).
3. For each `taqti-token` in tokens:
  a. Apply Arud rules (Algorithms 3-12, and additional rules if needed).
4. For each `taqti-token` in tokens:
  a. Call `assign code()` (Apply Algorithm 15).
5. For each sequence in the original sequence:
  a. Generate sequences (`Patterns`) for all possible verse patterns.
  b. For each pattern in the generated patterns:
    i. For each state function:
      - Apply Algorithm 16 (Match sequence).
6. Display the detected meter.
End
```

V. Example Test Case: Verse Step by Step Implementation

After explaining the step by step working of proposed methods, now, we run our solution for a test input to elaborate the UPMVM process of meter identification. Through this instrument, the verse writer can discern the metrical structure of a given verse. Looking closely at the steps that were taken in the implementation phase, Fig. 4 gives detailed view about how the second phase of the UPMVM is put into consideration. The second phase is explored with a verse example and divided into five sub-phases, each of which applies several functions to complete the specific tasks. In the second phase, 11 steps are summed up into five sub-phases i.e. Tokenization, orthography, syllable identification, weight assignment, and meter detection. Let consider a sample verse as a test case 1:

اچھا ہے دل کے ساتھ رہے پاسبان عقل

Once a verse is input to the system, the verse goes through the sub-phases of the phase 2 of UMPVM as shown.

1. **Tokenization (step 1-2)**

   In the first phase of pre-processing tokenization is performed as described above. The possible tokens of the above verse are as follows:

ع ق ل	پاسبان	رہے	ساتھ	کے	دل	ہے	اچ ھا

2. **Orthography (step 3 – 4)**

   Orthography is performed on the tokenized verse. Here, the scansion rules are applied where applicable to the identified token. The example output is as follows:

[	عقل[	پاسبان	رہے	ساتھ	کے	دل	ہے	اچھا	]

3. **Syllable Identification (Right to Left) (step 5 – 6)**

   The third phase transforms detected tokens into Arud tokens by applying Arud rules, which are applicable on each identified token. The output of syllable identification is shown below, where ھ in token اھچا is removed and, due to tashdid on ھچ, letter چ is counted twice. ھ is also removed from word ھتاس transforming it into تاس . The output of this phase is the Arud token which gives the sense of short, flexible, or long syllables:

[ل	عق	ن	با	س	ا پ	ہے ر	ت سا	ك ے	د ل	ے , ہ	چا	اچ]

4. **Assigning weights (Left to Right) (step 7-8)**

   In this second to last phase of verse implementation, each Arud token is detected as a short, long, or flexible syllable. Where the short syllable is assigned a weight of 1, the long syllable as 2, and the third flexible syllable as X. The example verse is assigned weights as shown below. This pattern is further used for meter matching.

   $$2\ 2\ x\ 2\ x\ 2\ 1\ 1\ 2\ 2\ 1\ 2\ 1\ 2\ 1$$

5. **Meter Matching (Left to Right) (step 9 – 10)**

   In this last phase of implementation, where verse meter is detected with its *state* function, each state function is matched to the above-required pattern in the metric function. These state functions are created using combined optimized

   DFA state sequences of identified meters. Each state function starts matching from state function q0, then according to the sequence generated above that calls the corresponding state

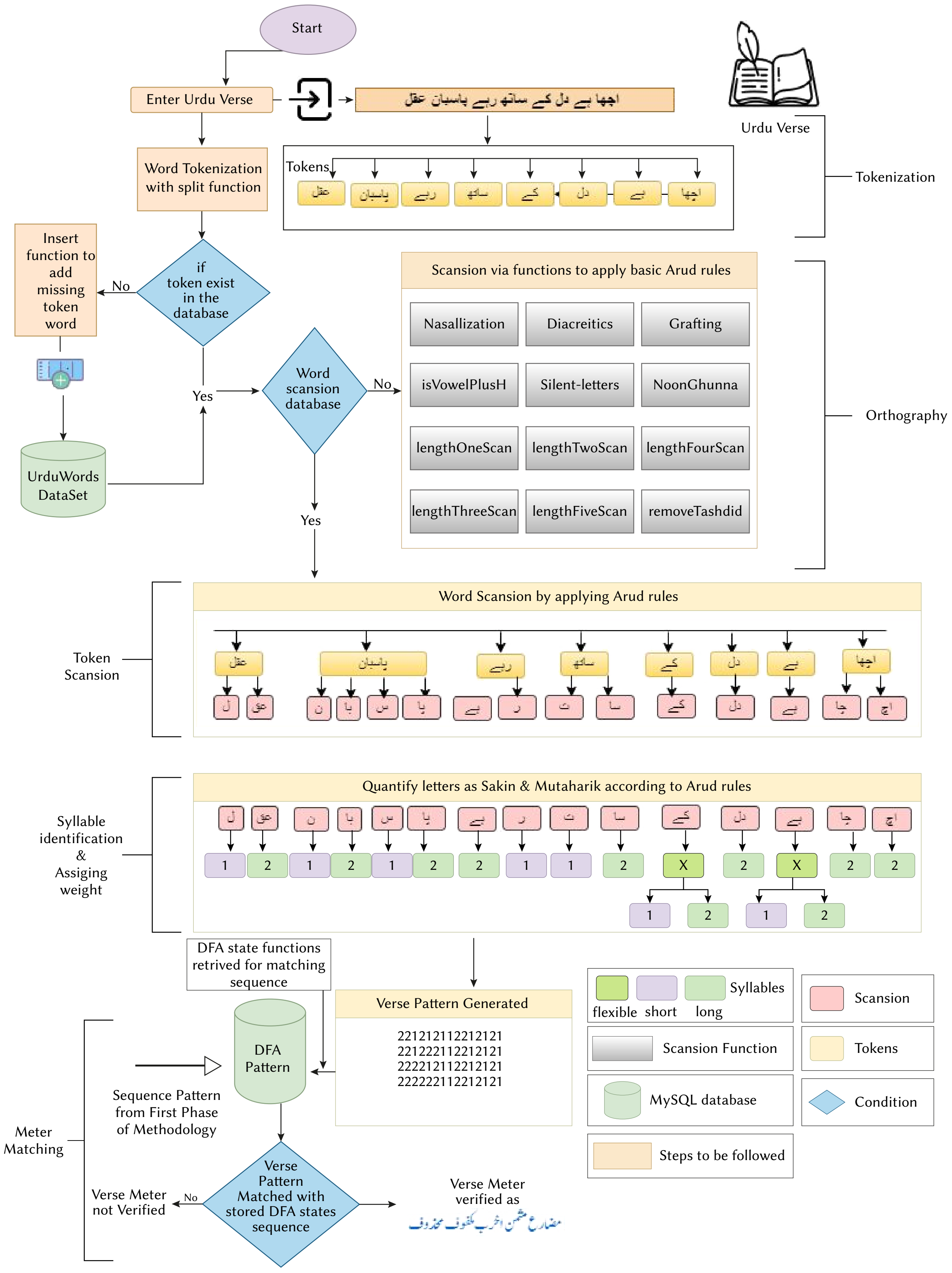
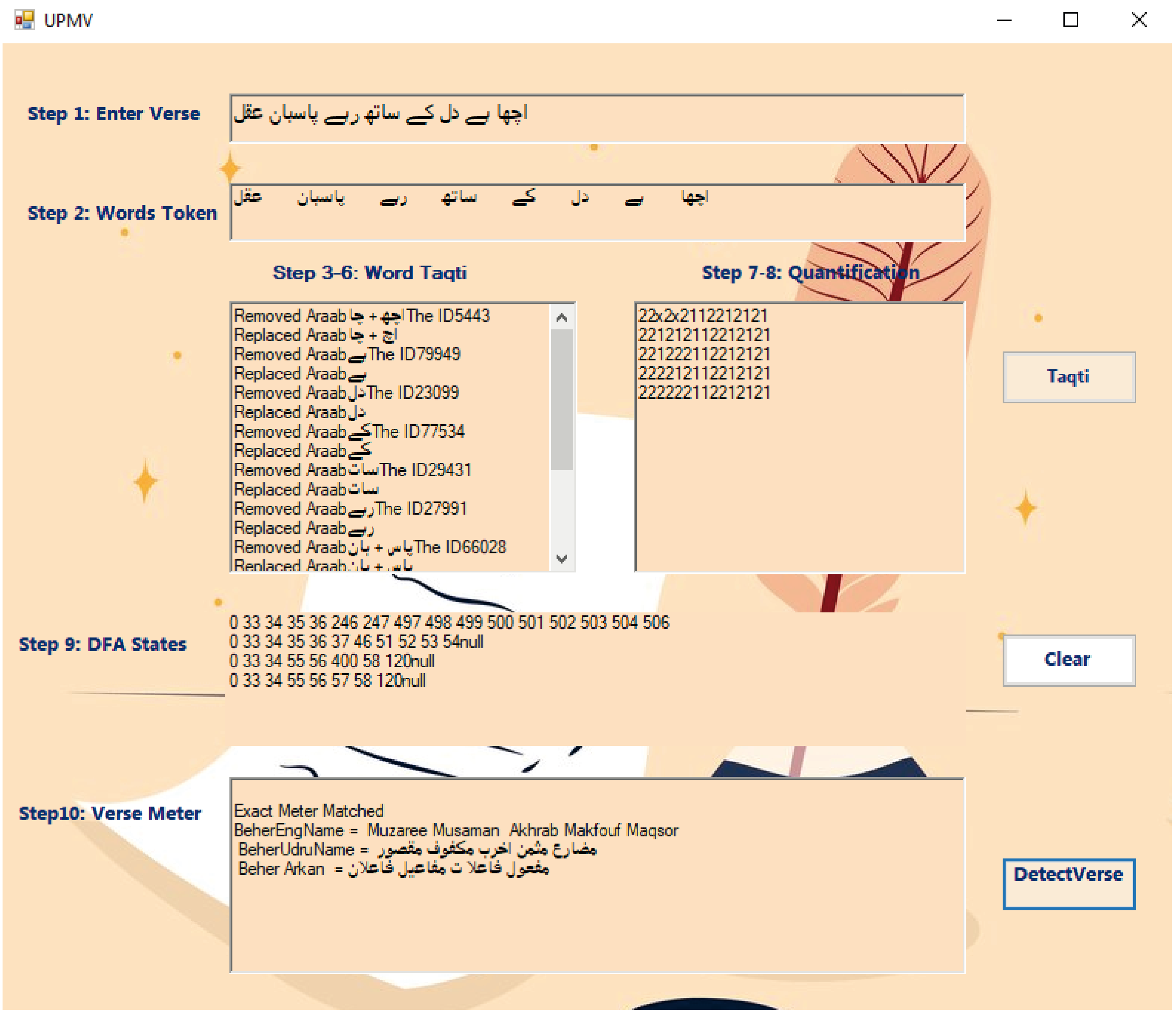


Fig. 4. Implementation Example of UPMVM.



اچھا ہے دل کے ساتھ رہے پاسبان عقل

اچھا ہے دل کے ساتھ رہے پاسبان عقل

```
Removed Araab چا + اچھThe ID5443
Replaced Araab چا + اچ
Removed Araab ہےThe ID79949
Replaced Araab ہے
Removed Araab دلThe ID23099
Replaced Araab دل
Removed Araab کےThe ID77534
Replaced Araab کے
Removed Araab ساتThe ID29431
Replaced Araab سات
Removed Araab رہےThe ID27991
Replaced Araab رہے
Removed Araab پاس + بانThe ID66028
Replaced Araab پاس + بان
```

```
22x2x2112212121
22121211212121
22122211212121
22221211212121
22222211212121
```

```
0 33 34 35 36 246 247 497 498 499 500 501 502 503 504 506
0 33 34 35 36 37 46 51 52 53 54null
0 33 34 55 56 400 58 120null
0 33 34 55 56 57 58 120null
```

```
Exact Meter Matched
BeherEngName =  Muzaree Musaman  Akhrab Makfouf Maqsor
 BeherUdruName =  مضارع مثمن اخرب مکفوف مقصور
 Beher Arkan  = مفعول فاعلا ت مفاعیل فاعلان
```

Fig. 5. UPMV System Screenshot.

functions until the end of the Tc pattern. In our test case, fq0() calls fq33() because the first sequence is 2, which invokes state function fq33(), and then will call state functions according to the generated sequence in previous sub-phase. The above verse will sequentially call the following functions one by one *[0-33-34-35-36-246-247-497-498-499-500-501-503-504-506]* to match the exact meter of the given verse until it reaches the terminal state. Here, algorithm 16 is implemented, to call the DFA state functions. The verse pattern generates four possible sequences and successively matches them. Every generated pattern as shown in step 8 calls the state functions to be matched with the stored DFA sequence and display the matched verse as shown in Fig. 5. The detected meter of test case 1 is [مکفوف مقصور مضارع مثمن اخرب]. Fig 5 shows all steps results of the UPMV tool.

VI. Evaluation Results

To measure the effectiveness of UPMV system, we use a (UD3) benchmark dataset. This labelled data is filtered out through two autonomous systems i.e. the Aruuz system and our UPMV tool for meter detection [34] system [32] is one of the commonly known and frequently used platforms for verse meter detection. Techniques such as cross-validation help maximize the utility of a limited dataset by iterative training and testing the model on different subsets of the data. We assess the outcomes of the Aruuz system by contrasting them with the findings of the UPMV through the utilization of various evaluation metrics.

Fig. 6 describes that in the model evaluation experiment we filtered UD3 from both autonomous systems UPMV and Aruuz. We carefully compared and evaluated the outcomes of both tools R1 and R2 with labelled datasets using contingency metrics. To evaluate the performance of our system, four well-known metrics are used, namely, (i) precision, (ii) recall, (iii) F1-measure [40], and (iv) accuracy. After executing all 500 test cases from both systems we analyzed that out of 20 basic metrics, the dataset covers 9 most frequently used Urdu meters. Most of the results show these nine meters as shown in Table IX.

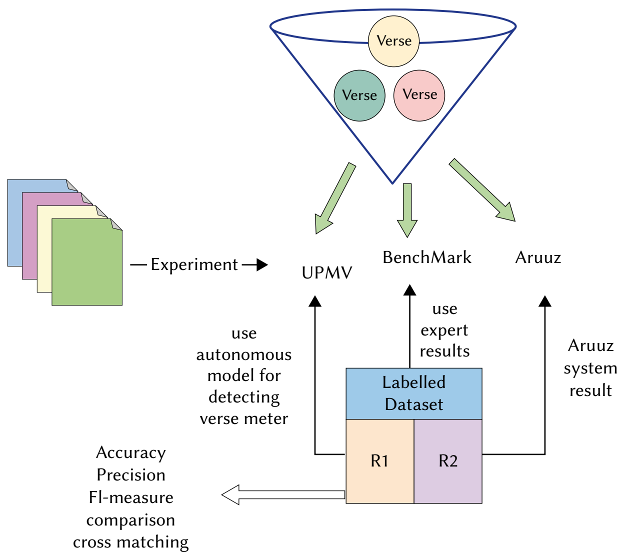


Fig. 6. Result Evaluation.

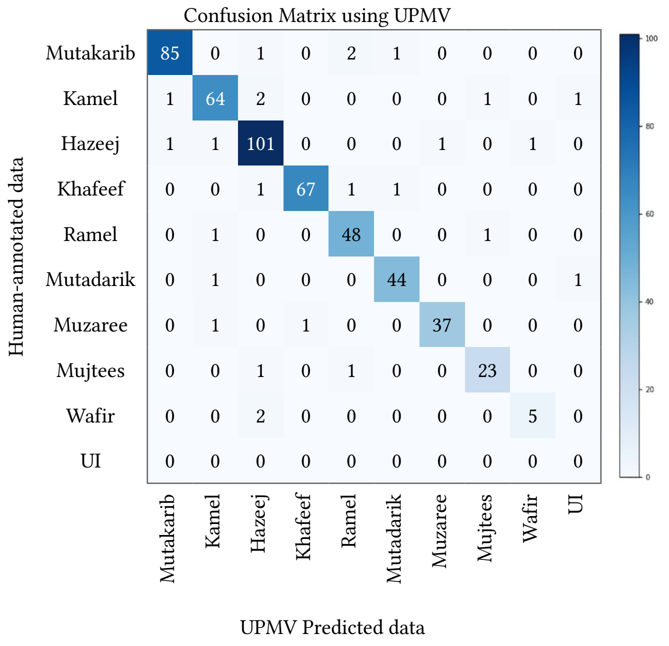
TABLE IX. No of Meter Collection in UD3

Meter	Description	No of Verse
Mutaqarb	متقارب	89
Kamel	کامل	69
Hazeej	ہزج	105
Khafef	خفیف	77
Ramel	رمل	50
Mutadarik	متدارک	46
Muzareeh	مضارع	39
Wafeer	وافر	5
Mujtees	مجتث	20

These nine meters are incorporated into our model evaluation. In Table IX the minimum count is for the Wafer وافر meter, 20 verses i.e. due limit use in Urdu poetry, and the maximum is for the Hazeej ہزج meter, 105 verses. Hazeej meter is the most frequently used meter in Urdu language having 48 sub-meters mentioned in UD1. Its number of sub-meter is relatively high as compared to all other Arud meters. Table IX presents a selection of meters from Urdu poetry literature, highlighting that **Hazeej (ہزج)** is the most frequently used meter, while **Wafer (وافر) is the least one.** The outcomes of the UPMV system were assessed and contrasted with the results obtained from the Aruuz system. The definitive value is attributed to human-annotated values, which are obtained from books that have been subjected to extensive evaluation by a wide list of experts prior to publication. We are quantifying the number of positive cases that can be predicted in order to validate the verse meter name and its feet/Afail افاعیل . Specifically, we are assessing the dependability of our model by determining how many of the correct predictions are positive cases. We use a confusion matrix, also known as a contingency table, which is a two-dimensional array that is employed to visually depict and represent the correlation between instances annotated by humans and those predicted by machines. The instances that were labelled by humans are displayed in each row, while the machine-predicted labels are indicated in each column.

In our case, since we are dealing with a multi-class classification problem (different verse meters), the concepts of true negatives and false negative are not directly applicable. True negatives refer to the cases where the model correctly predicts the absence of a particular class, but in our scenario, there is no "negative" class. We are interested in predicting the specific meter of each verse, not whether it belongs to a "negative" category. Confusion matrix contains TP, FP, TN and FN values. In our case we consider TP, FP, and FN, which are described as:

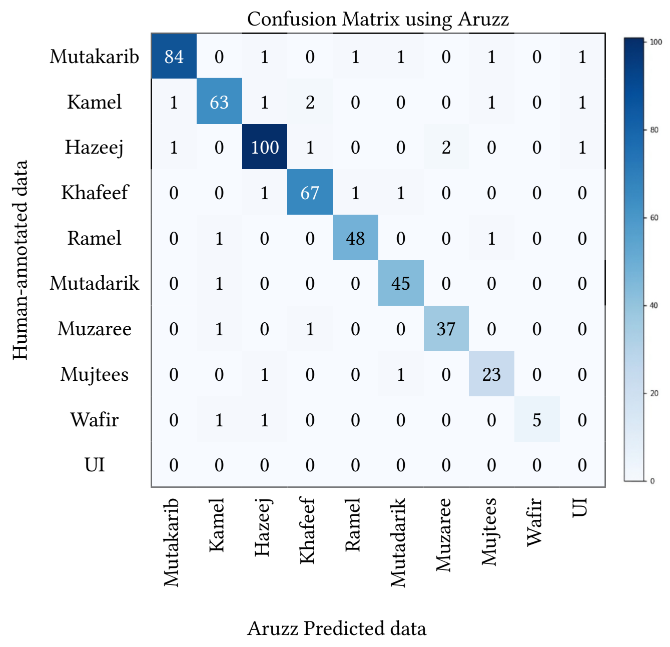
- True Positive (TP): These are the cases where the model correctly predicts the meter of a verse. In other words, if the model predicts that a verse belongs to a certain meter category, and the actual label of the verse matches that prediction, it is a true positive.
- False Positive (FP): These are the cases where our model predicts a verse to belong to a certain meter category, but the actual label of the verse does not match that prediction. In other words, the model incorrectly assigns a meter to a verse that does not belong to that category.
- False negative (FN): These are the cases where our model fails to predict the correct meter of a verse. If the actual label of a verse indicates that it belongs to a certain meter category, but the model predicts a different meter (or fails to predict any meter), it is a false negative. Unidentified (UI) verses are FN values.
- In this context, we have compiled a 10 x10 matrix of frequently used nine key Urdu poetry meters and an additional field of Unidentified (UI) verses to evaluate the performance of UPMV on a benchmark collection of verses. We compiled separate heat maps and contingency tables for Aruuz as well as the proposed UPMV framework on the same benchmark collection to assess the effectiveness and relatedness of the proposed system. The results of a multiclass classification task covering ten distinct classes are displayed in the confusion matrix shown in Fig. 7. The genuine class is denoted by each row, whereas the predicted class is identified in each column. The number of occurrences in which the predicted class is accurate is denoted by the cell values. In Fig. 7 and Fig. 8 the diagonal elements of the matrix represent correct predictions TP, where the true class matches the predicted class. Off-diagonal elements indicate misclassifications.


Confusion Matrix using UPMV

	Mutakarib	Kamel	Hazeej	Khafeef	Ramel	Mutadarik	Muzaree	Mujtees	Wafir	UI
Mutakarib	85	0	1	0	2	1	0	0	0	0
Kamel	1	64	2	0	0	0	0	1	0	1
Hazeej	1	1	101	0	0	0	1	0	1	0
Khafeef	0	0	1	67	1	1	0	0	0	0
Ramel	0	1	0	0	48	0	0	1	0	0
Mutadarik	0	1	0	0	0	44	0	0	0	1
Muzaree	0	1	0	1	0	0	37	0	0	0
Mujtees	0	0	1	0	1	0	0	23	0	0
Wafir	0	0	2	0	0	0	0	0	5	0
UI	0	0	0	0	0	0	0	0	0	0

Fig. 7. R1 Confusion Matrix using UPMV.

The confusion matrix shown in Fig. 7 describes diagonal true positives for each meter, with some verses being false negatives and misclassified, obtained with UPMV method (R1 results). Although we find some mismatched and UI meters the percentage of these is quite low. The analysis of heat-map shows that more than 250 verses lie in top first three meters. The first three meters Mutakarib متقارب, Kameel کامل and Hazeej ہزج meters have higher degrees of mismatch due to their large number of sub-meters. The remaining 6 meters had a lower degree of mismatching meters. The occurrence of false negative values in Urdu poetry is due to factors like poet's diacritics to maintain the verse's rhythm. Different factors/Affail افاعیل having the same weight detect different meters. The UPMV is a rule-based model, that is why it ignores rhythmic analysis, and displays exact match of the verse meter. It follows the rules and quantifies the syllables according to pre defined Arud rule. The same data is processed by the Aruuz system to get results R2 shown in Fig. 8.



	Mutakarib	Kamel	Hazeej	Khafeef	Ramel	Mutadarik	Muzaree	Mujtees	Wafir	UI
Mutakarib	84	0	1	0	1	1	0	1	0	1
Kamel	1	63	1	2	0	0	0	1	0	1
Hazeej	1	0	100	1	0	0	2	0	0	1
Khafeef	0	0	1	67	1	1	0	0	0	0
Ramel	0	1	0	0	48	0	0	1	0	0
Mutadarik	0	1	0	0	0	45	0	0	0	0
Muzaree	0	1	0	1	0	0	37	0	0	0
Mujtees	0	0	1	0	0	1	0	23	0	0
Wafir	0	1	1	0	0	0	0	0	5	0
UI	0	0	0	0	0	0	0	0	0	0

Fig. 8. R2 Confusion Matrix using Aruuz.

The Aruuz system was used to filter the same UD3 of 500 verses, to accurately identify exact meter names in each verse. However, some verses were misclassified as false negative values due to the complexity of the system's implementation. The Aruuz system uses different fuzzy logic to preserve the rhythmic structure of the verse, preventing misclassification. The heat-maps in Fig. 7 and Fig. 8 display the actual and predicted values, while the confusion matrix is used to assess performance using various measures. The measuring techniques used to compare the results are described below.

A. Precision

Precision is a quantitative measure that provides insight into the accuracy and reliability of positive forecasts. In our context, a high level of precision signifies that a large number of verses are marked correctly. It indicates that the high ratio of retrieved items is relevant and vice versa.

$$Precision = \frac{[TP]}{[TP + FP]} \tag{7}$$

In our case

$$Precision = \frac{[TP]}{[Total\ predictive\ positive]} \tag{8}$$

B. Recall

The total number of real positive cases accurately anticipated is known as recall. Another name for this metric is sensitivity. A lower recall implies that a sizable portion of positive examples are being missed by the model, whereas a greater recall shows that the model is more adept at catching positive occurrences. In our case, recall is defined as the number of true positives divided by the total number of verses that belong to the positive class.

$$Recall\ = \frac{[TP]}{[TP + FN]} \tag{9}$$

$$Recall = \frac{[TP]}{[Actual\ positive]} \tag{10}$$

C. F1-Measure

F1- measure is a single measure of performance of the test. A high F1_ Measure indicates that the model has both high precision and high recall, which is desirable for many classification tasks. It is beneficial when you have uneven class distribution.

$$F1 - Measure = \frac{[2 * Precision * Recall\ ]}{(Precision + Recall)} \tag{11}$$

D. Accuracy Rate

The accuracy rate (AR) for UPMV model and Aruuz system is computed as follows:

$$AR\ = \frac{(\ number\ of\ correctly\ identified\ verses\ x\ 100)}{(total\ number\ of\ dataset\ Verses)} \tag{12}$$

$$AR\ (UPMV) = \frac{474 * 100}{500} = 94.8$$

$$AR(Aruuz) = \frac{472 * 100}{500} = 94.4$$

A comparison was made between the results of the UPMV model and the Aruuz system using human annotated verses in UD3. The comparison was made between the obtained results with respect to F-score (Eq. 11), Precision (Eq. 8), and Recall (Eq. 10) [40].

The complete details of the model performance are represented in Table X. It shows the values of precision, recall, F1-measure, rates for each unique class using UPMV and Aruuz tools on UD3 verses. The findings were derived and highlight the inherent tradeoff between precision and accuracy. UMPV Mutakarib متقارب and Muzaree مضارع meters show the highest precision of 97%. The low performance is demonstrated by the wafer meter with 92% precision. Similarly, with the Aruuz method the highest precision was found in Majtees مجتث and Mutakarib متقارب, 100 and 97 respectively. The results of the AR analysis demonstrate that both the UPMVM and Aruuz models exhibit similar accuracy, with only minor fractional differences. However, UPMVM outperforms the Aruuz system despite Aruuz's 13 years of existence and rigorous validation processes. UPMVM delivers satisfactory performance in detecting the exact match of verse meters and shows potential for further improvement if additional test cases covering all meter types are incorporated. The findings suggest that the accuracy of the proposed model in Urdu meter classification can be enhanced by expanding the size of the dataset. UPMVM represents a significant advancement over the Aruuz system, as it addresses a notable limitation in the Aruuz database, which omits one of the Urdu poetic feet, "فاعلان," in its meter verification. Additionally, while the Aruuz database includes comparisons for approximately 143 Urdu meters, the UPMVM dataset (UD1) extends this capability by storing 192 Urdu meters for comparison.

Although the proposed model is designed for Urdu poetry, the UPMVM's framework could potentially be adapted to handle poetic meters in other languages with similar metrical systems, such as Persian, Punjabi, Othman or Arabic. This adaptation would further test its scalability in multilingual and multicultural contexts. The UPMVM

TABLE X: Comparative Evaluation of UPMV System With Aruuz

		UPMV -Evaluation R1			Aruuz -Evaluation R2		
S#	Meter Name	Recall	Precision	F1-Measure	Recall	Precision	F1- Measure
1	Mtمتقارب	95.5	97.7	96.59	94.38	97.38	95.86
2	Klکامل	94.12	94.4	94.26	91.56	94.02	92.77
3	Hjہزج	96.19	93.51	94.83	95.03	95.2	95.11
4	Khخفیف	94.73	96.18	95.45	94.72	93.72	94.22
5	Rm رمل	96	92.3	94.11	96	96	96.00
6	Mkمتدارک	95.65	95.65	95.65	97.82	93.7	95.72
7	Mzمضارع	94.8	97.3	96.03	94.23	94.87	94.55
8	Mjمجتث	90	94.73	92.30	90	100	94.74
9	Wfوافر	92	92.23	92.11	92	88.46	90.20
Average		**94.33**	**94.78**	**94.57**	**93.97**	**94.82**	**94.35**

model serves as a comprehensive and widely applicable framework for the analysis of Urdu poetry. Incorporating a user feedback mechanism enables users to identify ambiguous results, facilitating the model's evolution over time. The scalability of the UPMVM is demonstrated by its extensive dataset and capacity to accommodate a wider variety of poetic forms.

VII. Conclusion

The rapid digital change of modern life has led to a widespread desire for quick fixes and simple solutions to a variety of problems. This phenomenon is present in a number of fields, such as Urdu poetry, where prospective learners look for quick and easy access to educational materials. Because specialists are now faced with time constraints, there is less need for conventional mentorship, which was once used to ensure that poetry followed complex Arud norms.

As a result, an increasing dependence on Artificial Intelligence (AI) platforms has emerged to satisfy the growing need for comprehensive language acquisition. In order to address the gap between digital accessibility and human expertise in Urdu poetry, our study introduces the Urdu Poetry Meter Verification (UPMVM) framework. Our approach involved implementing a rule-based system in order to assess the viability of digitizing human expertise. The UPMVM consists of two phases: pre-processing Urdu poetry metrics collection and constructing their DFA for tracing pattern of each identified meter, and implementing the model to verify meters. The first phase focuses on pre-processing Urdu poetry metrics and assembling existing literature on the Arud system onto a single soft-form platform. The second phase involves algorithmic implementation, accepting a single verse without diacritics, and implementing seventeen different procedures to identify rhythmic patterns in Urdu poetry. By focusing on a dataset consisting of 500 Urdu verses, our UPMV model attained a remarkable 94% accuracy rate. The UPMVM precision, recall, and accuracy results were satisfactory. Our analysis surpasses simple rhythm verification by thoroughly quantifying every verse token in order to differentiate between "mutaharik" and "sakin" components. Nevertheless, obstacles continue to exist, especially when the machine is confronted with the ambiguity caused when distinct "afail" labels assigned equal weights, which occasionally leads to misclassifications. In contrast to conventional methodologies, our novel approach not only authenticates the adherence of verses but also offers comprehensive insights into their metrical composition, thereby enabling a more profound comprehension of metrics in Urdu poetry. Future research will focus on integrating rhythmic analysis into **UPMV** to enhance precision. Expanding the model's ability to process additional diacritics can further improve accuracy. Additionally, leveraging **machine learning** and **deep learning** techniques will optimize the effectiveness of **UPMVM**.

Acknowledgement

This work was supported by the Deanship of Scientific Research, Vice Presidency for Graduate Studies and Scientific Research, King Faisal University, Saudi Arabia, under Project GRANTS: KFU253418.

References

[1] M. Dar, "Authorship attribution in Urdu poetry," *Information Technology University (ITU), Lahore*, 2020.

[2] Z. Sherif, M. Eladawy, M. Ismail, and H. Keshk. "A Proposed System for the Identification of Modem Arabic Poetry Meters (IMAP)," In *15th International Conference on Computer Engineering and Systems (ICCES)*, pp. 1-5. IEEE, 2020.

[3] M. OajLakhnawii, "Mikyass-ul – Ashaar," Prosody book, 1875.

[4] S. Alam Sarwer, "Kitab-r-Arooz," 2008.

[5] M. Najmal-ul-Ghani, "Bahr-ul-Fasahet," Tayaba Noor printers, Lahore, 2018.

[6] P. W. Frances, and A. Khaliq, "*Urdu Meter: A Practical Handbook*," FW Pritchett and KA Khaliq, 1987.

[7] G.D. Pybus, "A Text-book of Urdu Prosody and Rhetoric," Baptist Mission Press, 1924.

[8] N. Balghi, "Tafheem ul Arud," Behar Afseet press, Patna, 1985.

[9] A. Jabbar, S. Iqbal, M.U. Khan, "Analysis and development of resources for Urdu text stemming," Language and Technology, vol. 1, no. 1, pp. 40-5, 2016.

[10] J. Abdul, S. ul Islam, S. Hussain, A. Akhunzada, and M. Ilahi, "A comparative review of Urdu stemmers: Approaches and challenges," *Computer Science, vol.* 34, 100195, 2019.

[11] A.A. Raza, A. Habib, J. Ashraf, M. Javed, "*A review on Urdu language parsing,*" International Journal of Advanced Computer Science and Applications, vol. 8, no. 4, pp. 93-97, 2017.

[12] N. Tariq, I. Ejaz, M.K. Malik, Z. Nawaz, and F. Bukhari, "Identification of Urdu ghazal poets using SVM," *Mehran University Research Journal of Engineering & Technology*, vol. 38, no. 4, pp. 935-944, 2019.

[13] AA. Raza, A. Athar, and S. Nadeem, "N-gram based authorship attribution in Urdu poetry," In *Proceedings of the Conference on Language & Technology*, pp. 88-93, 2009.

[14] R. Shahid, and Z.A. Qureshi. "Exploratory data analysis of urdu poetry," *arXiv preprint arXiv:2112.02145*, 2021.

[15] M.A. Alnagdawi, H. Rashideh, and F. Aburumman, "Finding Arabic poem meter using context free grammar," *Journal of Communication and Computer Engineering, vol.* 3, no. 1, pp. 52-59, 2013.

[16] R. Shalabi, G. Kana'an, and A. AL-Jarah, "Computing System for

Analyzing Arabic Poems Meter," *Yarmouk Research, Yarmouk University*, 2003.

[17] O. Alsharif, D. Alshamaa, N. Ghneim, "Emotion classification in Arabic poetry using machine learning," International Journal of Computer Applications, vol. 65, no. 16, 2013.

[18] M.Y. Dahab, A. AlAmri, B. Bagasi, E. AlMalki, O. AlBeshri, "Automatic Identifying Rhythm of Arabic Poem, " International Journal of Computer Applications, vol. 975, 8887, 2016.

[19] K. Baïna, H. Moutassaref, "An efficient lightweight algorithm for automatic meters' identification and error management in Arabic poetry," In Proceedings of the 13th International Conference on Intelligent Systems: Theories and Applications, pp. 1-6, 2020.

[20] B. Abuata, A. Al-Omari, "A rule-based algorithm for the detection of Arud meter in CLASSICAL Arabic poetry," The International Arab Journal of Information Technology, vol. 15, no. 4, pp. 661-667, 2018.

[21] A.I. Omer, M. P. Oakes, "Arud, the metrical system of Arabic poetry, as a feature set for authorship attribution," IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), IEEE, pp. 431-436, 2017.

[22] F. Alqasemi, A.H. Salah, N.A. Abdu, B. Al-Helali, G. Al-Gaphari, "Arabic poetry meter categorization using machine learning based on customized feature extraction," International Conference on Intelligent Technology, System and Service for Internet of Everything (ITSS-IoE), pp. 1-4, 2021.

[23] A. M. Mutawa, and A. Ayshah, "Determining the Meter of Classical Arabic Poetry Using Deep Learning: A Performance Analysis," Frontiers in Artificial Intelligence, vol. 8, 1523336, 2025.

[24] W.A. Yousef, O.M. Ibrahime, T.M. Madbouly, M.A. Mahmoud, "Learning meters of Arabic and English poems with Recurrent Neural Networks: a step forward for language understanding and synthesis," arXiv preprint arXiv:1905.05700, 2019.

[25] S. Zeyada, M. Eladawy, M. Ismail, H. Keshk, "A Proposed System for the Identification of Modem Arabic Poetry Meters (IMAP)," In 2020 15th International Conference on Computer Engineering and Systems (ICCES), IEEE, pp. 1-5, 2020.

[26] M. R. Abbas, K. H. Asif, "Computing prosody to detect the Arud meter in Punjabi Ghazal," Sādhanā, vol. 45, 246, 2020.

[27] A. Kaushal, K. Dutta, "Analysis of Performance Metrics for Classification of Punjabi Poetry using Machine Learning Techniques," In 2023 International Conference on Artificial Intelligence and Smart Communication (AISC), IEEE, pp. 680-684, 2023.

[28] A. Kurt, M. Kara, "An algorithm for the detection and analysis of arud meter in Diwan poetry," Turkish Journal of Electrical Engineering and Computer Sciences, vol. 20, no. 6, pp. 948-963, 2012.

[29] R. Shalabi, G. Kana'an, A. AL-Jarah, "Computing System for Analyzing Arabic Poems Meter," Yarmouk Research, Yarmouk University, 2003.

[30] O. Alsharif, D. Alshamaa, N. Ghneim, "Emotion classification in Arabic poetry using machine learning," International Journal of Computer Applications, vol. 65, no. 16, 2013.

[31] A. K. Al-Talabani, "Automatic recognition of Arabic poetry meter from speech signal using long short-term memory and support vector machine," ARO-The Scientific Journal of Koya University, vol. 8, no. 1, pp. 50-54, 2020.

[32] Aruuz, *Meters List*, accessed April 10, 2022, https://aruuz.com/resources/meterslist.

[33] Rakhta, Meter detection, accessed April 12, 2024, https://www.rekhta.org/.

[34] S. Zeeshan, *Aruuz*, GitHub repository, accessed March 25, 2022, https://github.com/sayedzeeshan/Aruuz

[35] M. S. Al-Shaibani, Z. Alyafeai, I. Ahmad, "Meter classification of Arabic poems using deep bidirectional recurrent neural networks" vol. 136, pp. 1-7, 2020.

[36] A. Muztar, Doctoral thesis "Urdu ka Aruzi Nizam and Asri Takaza", June 2022.

[37] A. Daud, W. Khan, D. Che, "Urdu language processing: a survey," Artificial Intelligence Review, vol. 47, pp. 279-311, 2017.

[38] R. Russell, "Some problems of the treatment of Urdu metre," Journal of the Royal Asiatic Society, vol. 92, no. 1-2, pp. 48-58, 1960.

[39] JFLAP, *JFLAP: Interactive Automata Theory Tool*, accessed March 20, 2022, https://www.jflap.org.

[40] W. Ruibo, J. Li, "Bayes Test of Precision, Recall, and F1 Measure for Comparison of Two Natural Language Processing Models," In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, pp. 4135–4145, 2019.

Asia Zaman

Mrs. Asia Zaman obtained her Ph.D degree from Gomal University, Pakistan in 2024. She is currently serving as an Assistant Professor at the Department of Computing and Software Engineering, Faculty of Computing at Gomal University, D.I.Khan, Pakistan since 2021. She get MS degree in Software Engineering from the International Islamic University Islamabad, Pakistan in 2013. She served six years as a Lecturer at King Faisal University, KSA. Her research interests include Natural Language Processing, Speech Text Conversion, Pattern Matching, Machine Learning, Deep Learning, Computer Vision, Object Detection in Agriculture fields, Image processing in health, and IOT security.

Zia-Ud-Din

Dr. Ziauddin is Associate Professor at Institute of Computing and Information Technology, Gomal University, Dera Ismail Khan, Pakistan. He is teaching there for last 34 Years. He has done his Master degree in Computer Science with distinction in 1990 Whereas Ph.D. in Computer Science in 2008. He has experience in many areas of Computer Science with emphasis on Software Process Improvement and Machine Learning. He has supervised many Doctoral and Master Level Scholars. Besides teaching, he is also a painter and poet.

Dr. Sajid Iqbal

Department of Information Systems, College of Computer Science and Information Technology, King Faisal University, Hofuf, Saudi Arabia. Sajid Iqbal received the Ph.D. degree from the Department of Computer Science, University of Engineering and Technology, Lahore, Pakistan. He is currently an Assistant Professor with the Department of Information Systems, College of Computer. He has published more than 30 papers in local and international journals and conferences. His research interests include medical image analysis, natural language processing, and computer vision.

Asma S. Alshuhail

She received her M.S. (2015) and Ph.D. (2022) in Computer Science and Informatics from Cardiff University, United Kingdom. She works as an assistant professor at King Faisal University, Saudi Arabia. Her primary research interests encompass Information Security and privacy, Machine Learning, advancements in privacy for artificial intelligence applications, and the enhancement of data security and privacy through Natural Language Processing.