2. SURVEY OF LITERATURE

A systematic survey of literature is a scientific tool of considerable value which brings together a number of separately conducted studies to summarize, appraise and synthesize the results and implications of otherwise unmanageable quantities of research. The objective of this chapter is to summarize and review as systematically as possible the quality of the state of the art studies enriching the field of machine transliteration. The presented literature review is organized into two broad sections. The first part is a discursive section that discusses diverse transliteration encoding schemes for non Roman scripts of the world. The second section is a comprehensive survey of machine transliteration and CLIR systems encompassing non-Indian and Indian languages. A considerable part of the survey focuses on studies predominantly investigating proposed methods for Arabic diacritization and their restoration. Finally, the related work particular to Shahmukhi script is examined for its effectiveness and observed limitations.

2.1 Transliteration Encoding for Non-Roman Scripts

Transliteration encoding among different scripts can be traced to 126 years back. On October 6, 1876 American Library Association was established. The aim of the Association, in their resolution, was "to enable librarians to do their present work more easily and at less expense. According to the transliteration time line posted by University of Arizona Library\(^4\) and Wellisch\(^5\) the first transliteration encoding scheme was developed in 1885. The American Library Association (ALA) created a system for representing Cyrillic characters. This is a simple encoding scheme because diacritics are not used (e.g. zh, kh, tch, sh, shtch, ye [for jat], yu, ya) and reverse transliteration is not considered. In 1898 the Prussian Instructions (PI) was created, which used a system of transliteration based on the Croatian model with diacritics. In 1909 the ALA and British Library Association (BLA) accepted both the systems. [36]

---

\(^4\) More details are available at web http://intranet.library.arizona.edu/users/brewerm/sil/lib/transhist.html
2.1.1 ALA-LC Transliteration Schemes for Non-Roman Scripts

ALA-LC is a set of standards for Romanization or the representation of text of non-Roman writing systems using the Latin alphabet. This system is used to represent bibliographic names by North American libraries and the British Library for acquisitions since 1975. The 1997 edition of the Transliteration Schemes for Non-Roman Scripts called ALA-LC Romanization Tables included 54 Romanization and transliteration schemes approved by the Library of Congress and the American Library Association. These tables were developed for use where consistent transliteration of a non-Roman script (e.g. Thai, Arabic, Devanagari, Punjabi, Kannada etc.) into the Roman alphabet was needed for instance in the case of proper names, titles and terms for which no appropriate Roman script equivalent exist. For some scripts the process of transliteration involves imparting characteristics of the Roman script such as capitalization and word division. The Thai script is typical example of one that has neither of these characteristics. This publication covered more than 150 languages written in various non-Roman scripts. A new table for Judeo-Arabic Romanization was approved in 2011 [37].

2.1.2 ISO: 15919: 2001 and ISO/IEC 10646 Standards

International Organization for Standardization ISO released an international standard for Transliteration of Devanagari and related Indic scripts into Latin characters. This International Standard provides tables which enable the transliteration into Latin characters from text in Indic scripts. The tables provide for the Devanagari, Bengali (including the characters used for writing Assamese), Gujarati, Gurmukhi, Kannada, Malayalam, Oriya, Sinhala, Tamil, and Telugu scripts which are used in India, Nepal, Bangladesh and Sri Lanka. The Devanagari, Bengali, Gujarati, Gurmukhi, and Oriya scripts are North Indian scripts, and the Kannada, Malayalam, Tamil, and Telugu scripts are South Indian scripts. The Burmese, Khmer, Thai, Lao and Tibetan scripts which also share a common origin with the Indic scripts, and which are used predominantly in Myanmar, Cambodia, Thailand, Laos, Bhutan and the Tibetan Autonomous Region within China, are not covered by this International Standard. Other Indic scripts whose character repertoires are covered by the tables may also be transliterated using this International Standard. [38]
In 1991, the ISO Working Group responsible for ISO/IEC 10646 (JTC1/SC2/WG2) and the Unicode Consortium decided to create one universal standard for coding multilingual text. Since then, the ISO 10646 Working Group (SC2/WG2) and the Unicode Consortium have worked together very closely to extend the standard and to keep their respective versions synchronized. Although the character codes and encoding forms are synchronized between Unicode and ISO/IEC 10646, but they are not same because the Unicode Standard imposes additional constraints on implementations to ensure that they treat characters uniformly across platforms and applications. To this end, it supplies an extensive set of functional character specifications, character data, algorithms and substantial background material that is not present in ISO/IEC 10646. Each version of the Unicode Standard identifies the corresponding version of ISO/IEC 10646. For example, Unicode 5.0 has the same character repertoire as ISO/IEC 10646:2003 with Amendments 1 and 2 applied.

### 2.1.3 ITRANS Encoding [39]

The first most noteworthy work on transliteration from Roman (English) to Indian languages was implemented in ITRANS in early 1991. ITRANS is a representation of Indian language alphabet in terms of ASCII (http://www.aczoom.com/itrans/). There are typically about 13 to 18 vowels and 36 to 54 consonants in the Indian language [40] while there are only 26 letters in the English alphabet. Since Indian text is composed of syllabic units rather than individual alphabetic letters, ITRANS uses a combination of two or more letters of the English alphabet to represent an Indian language syllable. ITRANS uses non-alphabetic characters such as “[“, “\”, “”” in some of the syllables to accommodate multiple sounds in Indian languages corresponding to the same English letter [41].

There are two popular versions of ITRANS encoding known as ITRANS 5.1 and ITRANS 5.3. In general, ITRANS offers a choice of two input encodings: ITRANS 5.1 or 5.3 encoding, and the CS/CSX (Classical Sanskrit Extended) encoding. ITRANS encoding is a 7-bit ASCII encoding (English alphabet), while the CS/CSX encoding is an 8-bit encoding. The ITRANS encoding uses multi-character English codes to represent each Indic Script letter, while the CS/CSX encoding uses a one-character code to represent each Indic Script letter. The ITRANS 5.3 encoding table, shows many codes having multiple choices, for example ऋ → R^i / RRi and ऊ → L^i
/ LLi, which implies that you can use either "R^i" or "RRi" and "L^i" or "LLi". According to ITRANS 5.1 and ITRANS 5.3 schemes the transliteration encodings of word संस्कृत have three and six unique forms as shown in Table 2. The ITRANS 5.3 encoding scheme is an extension of ITRANS 5.1 scheme. The CSX scheme is a specific representation that was evolved for displaying Sanskrit Text using diacritics.

<table>
<thead>
<tr>
<th>संस्कृत</th>
<th>ITRANS 5.1</th>
<th>ITRANS 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>संस्कृत</td>
<td>R^i</td>
<td>R^i / RRi</td>
</tr>
<tr>
<td>संस्कृत</td>
<td>Sa.nskR^ita</td>
<td>sa.nskR^ita / sa.nskRRita</td>
</tr>
<tr>
<td>संस्कृत</td>
<td>saMskR^ita</td>
<td>saMskR^ita / saMskRRita</td>
</tr>
<tr>
<td>संस्कृत</td>
<td>Sa.mskR^ita</td>
<td>sa.mskR^ita / sa.mskRRita</td>
</tr>
</tbody>
</table>

2.1.4 IAST and NLK Romanization Scheme

International Alphabet of Sanskrit Transliteration (IAST) is the popular transliteration scheme for Romanization of Sanskrit and Pali. Since the late 18th century, Sanskrit has also been written with the Latin alphabet. The most commonly used system is the IAST, which was been the standard for academic work since 1912. IAST is based on a standard established by the International Congress of Orientalists at Geneva in 1894. With some exceptions, IAST can be seen as a subset of ISO 15919. IAST encoding is often used in printed publications, especially for religious books dealing with ancient Sanskrit and Pali. This transliteration scheme has an advantage that it allows a lossless Romanization of Indic scripts [42].

The National Library at Kolkata (NLK) Romanization scheme, intended for the Romanization of all Indic scripts, is an extension of IAST. In the National Library at Kolkata, Romanization is the most widely used transliteration scheme in dictionaries and grammars of Indic languages. The Romanization tables use Devanagari but include letters from Kannada, Tamil, Malayalam and Bengali to illustrate the transliteration of non-Devanagari characters [43].

2.1.5 Harvard-Kyoto Conversion

To represent online content of Sanskrit language and other languages based on Devanagari script the Harvard-Kyoto system of transliteration was created at Harvard University and the University of Kyoto. Harvard-Kyoto is very simple scheme with
the aim that people all around the world could represent nonstandard language
sounds in plain text without having to worry about having the right fonts installed. As
was meant for, it is predominantly used informally in e-mail, and for electronic texts
[44].

2.1.6 OM Transliteration: Unified Representation for Indian Languages

To overcome the drawbacks of ITRANS, Madhavi, et al. [41] have redesigned a
novel mapping scheme called OM, which has become no longer a transliteration
mechanism alone, but a platform over which many other Indian language
applications have been built. OM encoding uses the same representation both for
keyboard input and for display. It is similar to ITRANS which uses combinations of
the English alphabet to represent Indian syllables. Unlike ITRANS, it is case
independent and avoids excessive use of non alphabetic characters.

The key points are:
– Easy to remember: ITRANS encoded text is a large mixture of capital, small and
special letters that leave it highly difficult to read. In this scheme the English
alphabet combinations are designed in such a manner that they are easy to
remember and natural to read like English.
– It is case-insensitive mapping: while preserving readability, this feature allows the
use of standard natural language processing tools for parsing and information
retrieval to be directly applied to the Indian language texts.
– Phonetic mapping, as much as possible: this makes it easier for the user to
remember the key combinations for different Indian characters. ASCII
representation may be used simply as a means of typing the text with standard
keyboard.
– Separates the storage that is in ASCII and the rendering that is dependent on the
fonts chosen. This paves the way for a language independent universal
representation; a fact that had been exploited in search engines. [45,46].

For transliteration to Indian languages, OM representation is mapped to the Indian
language fonts for display or can be converted to any other format such as Unicode
or Acharya [40] wherever required.
2.1.7 ISCII and PC-ISCII

Indian Standard Code for Information Interchange (ISCII) is the character code for Indian languages that originated from Brahmi script. ISCII was evolved by the standardization committee under the Department of Electronics during 1986-88, and adopted by the Bureau of Indian Standards (BIS) in 1991 (IS 13194:1991). Unlike Unicode, ISCII is an 8-bit encoding that uses escape sequences (special attribute character) to announce the particular Indic script represented by a following coded character sequence. As shown in Table 3 the code points are assigned in the upper ASCII region (160-255) for the aksharas and matras (vowel extensions) of the language. Special characters are included to specify how a consonant in a syllable should be rendered. In an 8-bit environment, the lower 128 characters are the same as ASCII character set. The top 128 characters cater to all the ten Indian scripts based on the ancient Brahmi script. In a 7-bit environment the control code SI can be used for invocation of ISCII code set, the control code SO can be used for reselection of the ASCII code set.

There is also a version of this table known as PC-ISCII. In this version, the first three columns of the ISCII-91 table have been shifted to the starting location of 128. PC-ISCII has been used in many applications based on the GIST Card, a hardware adapter which supported Indian language applications on an IBM PC.

This is the very first attempt at syllable level coding of Indian languages and it is not well accepted due to following reasons[40],[47]:

− The multiple byte representation for the syllables of Indian languages may vary from 1 byte to 10 bytes for a syllable and this poses hurdles in linguistic processing.

− The text in ISCII may be displayed using many different fonts for the same script. This needs specific rendering software which can map the ISCII codes to the glyphs in a matching font. Multi-byte syllables will have to be mapped into multiple glyphs in a font with a language dependent manner.

− In ISCII table, the nukta symbol ◌ is not a matra but a diacritic mark. When a nukta is attached to a consonant it yields a derived consonant or another consonant. To preserve the sorting order, it was kept following the matra symbols and not with the other diacritic marks. Further, ISCII code has compromised in grouping the consonants of the languages into a common set that does not preserve the true
sorting order of the aksharas across the languages. Specifically, some aksharas of Tamil, Malayalam and Telugu are out of place in the assignment of codes.

− No place was provided for the code points corresponding to the frequent conjuncts to ensure applicability to all Indian scripts.

Table 3 ISCII Code Points

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>NUL</td>
<td>DLE</td>
<td>SP</td>
<td>@</td>
<td>P</td>
<td>`</td>
<td>p</td>
<td>Õ</td>
<td>Ï</td>
<td>Ò</td>
<td>Ó</td>
<td>Ô</td>
<td>Ô</td>
<td>Ô</td>
<td>Ô</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>&quot;</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>E</td>
<td>U</td>
<td>c</td>
<td>u</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>BEL</td>
<td>ETB</td>
<td>'</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>BS</td>
<td>CBN</td>
<td>(</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>10</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>VT</td>
<td>ESC</td>
<td>;</td>
<td>K</td>
<td>[</td>
<td>k</td>
<td>{</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td></td>
<td></td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
</tr>
<tr>
<td>D</td>
<td>13</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>M</td>
<td>]</td>
<td>m</td>
<td>}</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>14</td>
<td>SO</td>
<td>RS</td>
<td>&gt;</td>
<td>N</td>
<td>^</td>
<td>n</td>
<td>~</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>15</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>O</td>
<td>_</td>
<td>o</td>
<td>DEL</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td>Ñ</td>
<td></td>
</tr>
</tbody>
</table>

2.1.8 Unicode Standard

Unicode is the first attempt to produce a standard for multilingual documents. Unicode provides a unique number for every character, no matter what the platform is, no matter what the program it runs on and no matter what the language is. The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, Microsoft, Oracle, SAP, Sun, Sybase, Unisys and many others. Unicode is required by modern standards such as in XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML, etc. It is supported in many operating systems, all modern browsers, and many other products. Unicode 5.0 has the same character repertoire as ISO/IEC 10646:2003 with amendments 1 and 2 applied and Unicode 6.0 adds the repertoire from amendments 3 through 8.
The fundamental idea behind Unicode (and ISO/IEC 10646) is that a superset of characters from all the different languages/scripts of the world be formed so that a single coding scheme could effectively handle almost all the alphabet of all the languages. Unicode assignments may be viewed geometrically as a stack of planes, each plane having one and possibly multiple chunks of 128 consecutive code values. Logically related characters or symbols have been grouped together in Unicode to span one or more regions of 128 code values.

For Indian languages, Unicode consortium adopted the 1988 version of ISCII-8 as the base for the 16-bit Unicode for allocating codes to different Indic scripts. Although the consortium tried to preserve the basic characteristics of ISCII coding, ISCII differed significantly from Unicode [47]. Unlike ISCII, which has a uniform coding scheme for all the languages, Unicode has provided individual planes for the nine (Bengali, Devanagari/Hindi, Gujarati, Punjabi, Kannada, Malayalam, Oriya, Tamil, Telugu) major languages of India. Within individual language planes of 128 code values, each assignment are language specific though the ISCII base has been more or less retained. [48-50].

2.1.8.1 ISCII vs. Unicode

The ISCII design exploited commonality of the Indic scripts and allocated code points for the superset of the enhanced Devanagari symbols. The graphical or the compositional aspect of individual characters and the script is not a consideration in its design. Therefore, ISCII applies to all Indic scripts, which makes transliteration among Indic scripts a straightforward task. Unicode, however, is more oriented toward facilitating script composition. It does not reflect in any way what could be common features of a group of scripts that could be dealt with uniformly for text processing. Unicode assigned a separate page for each one of the scripts. Thus, as one perceives more compositional features in the scripts, the demand for including more and more symbols continues. In ISCII, however, the symbols relate to the articulatory aspect of the associated speech, and it remains constant as long as all the articulatory aspects have been considered [48-50].

2.1.8.1.1 General Differences

Except for a few minor differences, ISCII and Unicode correspond directly [48-50]. Unicode is designed to be a multilingual encoding that requires no escape sequences
or switching between scripts. For any given Indic script, the consonant and vowel letter codes of Unicode are based on ISCII. ISCII allowed control over character formation by combining letters with the characters Nukta, Inv, & Halant. Unicode provides similar control with the ZWJ (Zero Width Joiner) & ZWNJ (Zero Width Non Joiner) characters.

- There are four uses of Nukta in ISCII. Unicode uses the first two only. Unicode doesn't use nukta for soft halant and doesn't use it for code extension. Unicode does use nukta to represent the nukta diacritic either in cases such as "ka" U+0958 or cases like "nnna" U+0929. Unicode doesn't use nukta for the "om" character (e.g. chandrabindu + nukta in ISCII, which is encoded as a separate character in Unicode).

- Unicode doesn't have an "invisible letter" (INV) like ISCII. One use of INV in ISCII is as a base letter that is used to express a space or no-break space (NBSP) in Unicode, depending on whether the result is to be a "word-like" character or not:

<table>
<thead>
<tr>
<th>ISCII</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>INV + vowel-sign</td>
<td>SPACE + vowel-sign</td>
</tr>
<tr>
<td>INV + vowel-sign</td>
<td>NBSP + vowel-sign</td>
</tr>
</tbody>
</table>

- The ISCII Attribute code (ATR) is not represented in the Unicode Standard, which is a plain text standard. The ISCII Attribute code is intended to explicitly define a font attribute applicable to following characters, and thus represents an embedded control for the kinds of font and style information which is not carried in a plain text encoding.

- The ISCII Extension code (EXT) is also not represented directly in the Unicode Standard. The Extension code is an escape mechanism, allowing the 8-bit ISCII standard to define an extended repertoire via an escaped re-encoding of certain byte values. Such a mechanism is not required in the Unicode Standard, which simply uses additional code points to encode any additional character repertoire.

- Soft halant and explicit halant in ISCII are represented as:
1) The "explicit halant" of ISCII: 2) The "soft halant" of ISCII:

<table>
<thead>
<tr>
<th>ISCII</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Halant + Halant</td>
<td>Halant + ZWNJ</td>
</tr>
<tr>
<td>Halant + Nukta</td>
<td>Halant + ZWJ</td>
</tr>
</tbody>
</table>

2.1.8.1.2 Issues with Gurmukhi Script

It is possible to convert Gurmukhi ISCII text to Unicode and back again without loss of information, but in some cases this will not result in readable Gurmukhi text [50]. There are several changes required in Gurmukhi reconversion as discussed below:

**Bindi and Tippi**

*Bindi* and *Tippi* are encoded using a single code point in ISCII (0xA2) and the underlying rendering engine selects the correct glyph. However, in Unicode they are given two separate code points. Thus, 0xA2 should be converted to *Tippi* (◌ੰ) when:

- The preceding letter is a consonant (ignoring any Nuktas)
- The preceding letter is any of the vowel sign like  ਰੀ,  ਡੀ or  ਠੀ.
- The preceding letter is Letter  ਅ or  ਇ.
- In all other cases, the sign should remain a *Bindi* (◌ੲ).

In reverse, when converting from Unicode to ISCII, both *Bindi* and *Tippi* should be converted to *Bindi* (0xA2).

**Consonant Clusters**

In general, consonant clusters are handled in the same manner in Unicode and ISCII. However, Unicode differs in that it encodes geminate consonants using a separate *Adhak* (◌ੱ) sign. As shown in Table 4, during conversion from Unicode to ISCII, an *Adhak* followed by a consonant in Unicode is converted in ISCII to a consonant, followed by *Halant* (0xE8) and then by the same consonant again.

<table>
<thead>
<tr>
<th>ISCII-91</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consonant + Halant (0xE8) + Consonant</td>
<td>Adhak (◌ੱ) + Consonant</td>
</tr>
</tbody>
</table>

| ਦ + ਏ + ਦ | ↔ | ਏ + ਦ |

Table 4 Encoding of Gurmukhi Consonant Clusters
Another difference is in the case of Rra Ṣ (U+0A5C). It is treated as a nukta character in ISCII but as a full character in Unicode.

2.2 Machine Transliteration Systems

2.2.1 Non-Indian Languages

This section highlights the research focusing on the transliteration systems between non-Indian languages. Transliterating names from Arabic into English in either direction is a difficult task, mainly due to the differences in their sound and writing systems. Arbabi et al. [27] at IBM presented Arabic to English transliteration system. They developed a hybrid neural network and knowledge-based system to generate multiple English spellings for Arabic personal names. The program first inserts the appropriate missing vowels into the Arabic name, then converts the name into a phonetic representation, and maps this representation into one or more possible Roman spellings of the name. The proposed approach is a hybrid application comprising a knowledge-based system (KBS) and artificial neural networks (ANNs). The KBS is constructed along linguistic vowelization rules which can be applied to a large class of unvowelized Arabic names. This class of “conforming” names follows strict rules of vowelization which are based on the number and structure of the root radicals and their relative positions with respect to other consonants and long vowels. An artificial neural network is used to filter those names from the input which otherwise would be vowelized inappropriately by the KBS. During evaluation they have used broad and narrow approaches. The broad approach covers closer to 80% of the names, but generates a much higher percentage of extraneous vowelizations. The narrow approach vowelizes over 45% of the names presented to it, with a 96.9% accuracy of vowelizations. This work is extendable to other dialects of Arabic or even to languages which use the Arabic alphabet but are fundamentally a different language like Persian, Pashto, Kurdish and Urdu.

Knight and Graehl [3], [21] described a back-transliteration system for Japanese. It comprised a generative model of how an English phrase becomes Japanese. They have presented a process of five stages to do back transliteration as:

1. An English phrase is written
2. A translator pronounces it in English
3. The pronunciation is modified to fit the Japanese sound inventory
4. The sounds are converted into the Japanese katakana alphabet
5. Katakana is written

They build statistical models for each of these five processes. A given model describes a mapping between sequences of type A and sequences of type B. These models were constructed automatically from data like text corpora and dictionaries. The accuracy of back transliteration was 27% and 64% by human subjects and machine respectively. When they use OCR, 7% of katakana tokens are misrecognized, affecting 50% of test strings, but translation accuracy drops from 64% to 52% only.

Later on, Stalls and Knight [22] adapted the same approach for back transliteration from Arabic to English of English names. Like Japanese, Arabic text frequently contains foreign names and technical terms that are translated phonetically. They observed that it is not trivial to write an algorithm for turning English letter sequences into Arabic letter sequences, and indeed, two human translators will often produce different Arabic versions of the same English phrase. There are many other factors that increase transliteration complexity such as some English vowels are dropped in Arabic writing, Arabic and English vowel inventories are also quite different, several English consonants have more than one possible Arabic rendering and even more, consonants like English D are sometimes dropped. Among other things, most important is that a human or machine translator must imagine sequences of dropped English vowels because Arabic is less deterministically phonetic; short vowels do not usually appear in written text. Long vowels, which are normally written in Arabic, often but not always correspond to English stressed vowels; they are also sometimes inserted in foreign words to help disambiguate pronunciation.

The first two models of Knight and Graehl, [3] deal with English only; the same was directly reused for Arabic/English transliteration. Due to lack of a large bilingual dictionary of names and terms for Arabic/English, they handcrafted a small 150-word dictionary. They supplied a list of 2800 test names in Arabic for testing and received translations for 900 of them. Those not translated were frequently not foreign names at all, so the program is right to fail in many such cases. The program offers many good translations but still makes errors of omission and commission. Some of these
errors show evidence of lexical or orthographic influence or of interference from other languages such as French.

**Wan and Verspoor** [52] developed a fully handcrafted rule-based transliteration system for **English to Chinese** proper nouns. The algorithm for auto transliteration involves five main stages: Semantic abstraction, Syllabification, Sub-syllable divisions, Mapping to Pinyin, and Mapping to Han Characters. The system first syllabifies English words based on a rule set and instance learning. The syllabification module identifies syllable boundaries based on consonant clusters and vowels. A sub-syllabification procedure then divides each identified syllable into the form of consonant-vowel, i.e. conforming to the monosyllabic nature of Chinese. The sub-syllables are then mapped to the pinyin equivalents by means of a handcrafted mapping table. The aim of the algorithm was to produce a transliteration understandable by reader of Chinese text. While this approach is adhoc and dialect dependant, the degree to which the transliteration is recognized by human speaker is dependent in part on the length of the original names that is, longer names with many syllables are less recognizable than shorter names.

**Kawtrakul et al.** [7] presented phoneme-based back transliteration between **Thai to English**. They used rule-based as well as statistical based learning approaches. Thai text retrieval system always involves documents that use loan words. They described an algorithmic approach to backward transliteration for improving Thai document retrieval process. They have also performed unknown word analysis and found that the average number of unknown words is about 15% of which 68.28% are loan words and 21.93% are foreign words. This signifies that automatic back transliteration has a great practical importance in Thai text retrieval and automatic indexing. They have proposed three serious problems in this back transliteration. Firstly, due to non-native accent, there are considerable variations in how a word may be transliterated into Thai from English. Secondly, they have faced loss of information because Thai orthography does not cover all sounds of English which is some what similar to our research work. The third problem realized in the official transliteration scheme maps between Thai and English was to represent the way English words are pronounced. Additionally, unlike the Japanese language which has special characters called Katakana [3], for writing loan word, and the Korean language that has a case marker called Josa and Emoni for identifying a loan word, Thai language has no orthographic method to distinguish loan words from native
Thai words. Further more, Thai sentences have no blank to mark each word within the same sentence. The difficulty of backward transliteration, then, includes the identification and extraction of a loan word. Therefore, transliteration process first performs the identification of loan/unknown words like word segmentation. Then each syllable-like unit is mapped to phonemes and finally, the most likely English word from Oxford Advanced Learners’ Dictionary is selected using Fuzzy matching technique. This model of back transliteration works by approximately 95% of accuracy including phonetic equivalent.

Darwish et al. [53] described hand-crafted grapheme-based system between English to Arabic. Each English letter was mapped to the closest sounding Arabic letter or letters. These mappings were decided manually. Most English letters were given a single Arabic equivalent; a few had more than one. Darwish’s system has the same purpose as Abduljaleel and Larkey [54]. However, the System of Abduljaleel and Larkey differs in that it learns the mappings automatically from the data itself.

The Buckwalter Arabic Transliteration was developed at Xerox by Tim Buckwalter [55, 56] for practical storage, display, and email transmission of Arabic text in the environment where the display of genuine Arabic character is not possible or convenient. Buckwalter system is a true or strict orthographical transliteration, as opposed to the majority of transcription of Arabic that conveys phonological or morphological information. It is a transliteration scheme that representing Arabic orthography strictly in one-to-one mapping, unlike the more common Romanization schemes that add further morphological information not expressed in Arabic script. Therefore in this transliteration an Arabic Character vav (ן) will be transliterated to “w” regardless of whether it is realized as a vowel or a consonant. Only when the vav (ן) is modified with hamza (ֲ) does the transliteration change to “&”. Text encoded in ASMO449 or ISO-8859-6 can be converted trivially and unambiguously into the Buckwalter transliteration and the reverse mapping is equally straightforward.

Al-Onaizan and Knight [4] presented a hybrid system between Arabic to English. The transliteration algorithm is based on sound and spelling mappings using finite state machines. The vowelization rules described by Arbabi et al. [27] apply only to Arabic names that conform to strict Arabic morphological rules. Any name that does not conform to the morphological rules is ignored and hence no transliteration will be attempted. This restriction limits the applicability of Arbabi et al. approach since many a person and organization names do not conform to morphological rules,
especially loan words and foreign names. Therefore, to investigate the suitability of their proposed models for transliteration, each name in the list is classified in one of three categories: ARABIC for names of Arabic origin, ENGLISH for names of English origin, and OTHER for names of other origins including Chinese, Russian, and Indian etc. They modeled pure phonetic based, pure spelling based and hybrid model by linear combination of the two models. They observed that the spelling based model is more accurate than state of the art phoneme-based model that can be easily trained. For experiments, they used exact matching Top1, Top20 criterion and human subjective evaluation when more that one possible transliteration is possible. Finally, the spelling-based model produced more accurate transliterations on all categories. But when top-20 transliterations were considered then the spelling based model was slightly less accurate.

**Jung et al.** [23] model is a statistical **English to Korean** transliteration model that generates transliteration candidates with probability. The model is designed to exploit various information sources by extending a conventional Markov window. They have described an alignment and syllabification of pronunciation units instead of a statistical method proposed for accurate and fast operation. For the evaluation, they constructed a training corpus of 8368 English-Korean word pairs. 90% of the corpus is used as training data and 10% of the corpus as test data. The proposed model demonstrates significant improvement in its performance. The experimental results show a recall of 0.939 for trained words and 0.875 for untrained words when the best or Top 10 candidates are considered.

**Oh and Choi** [57] discussed a **phoneme-based** transliteration model between **English and Korean** using pronunciation and contextual rules. Unlike the previous work of **Lee et al.** [8], **Kim et al.** [10] and **Kang et al.** [12], their system exploited phonetic information such as phoneme and its context and also used word formation information such as English words of Greek origin. This method shows significant improvements by a 31% gain in word accuracy. Initially, they performed English pronunciation unit (EPU) to phoneme alignment to find out most phonetically probable correspondence between an EPU and phoneme. The accuracy of EPU-to-Phoneme alignment is stated as 99%. To handle the complex word forms (those composed of two base nouns in P-DIC) they performed word segmentation. The next aspect is to detect English word with or without Greek origin from training data of G-class or E-class respectively to estimate their pronunciation. Their chunking EPU
module shows 91.7% precision. After estimating pronunciation, Phoneme to Korean conversion is performed. The comparison test with previous work shows that their system has highest 90.82% accuracy where the others like Lee et al. has 69.3%, Kim et al. has 79.0% and Kang et al. has 78.1% of accuracy.

**Goto et al.** [16] proposes a method of automatic transliteration from **English to Japanese** words. They modeled a **grapheme-based** approach that successfully transliterates an English word not registered in any bilingual or pronunciation dictionaries by converting each partial letters in the English word into Japanese katakana characters. In such transliteration, identical letters occurring in different English words must often be converted into different katakana. To produce an adequate transliteration, the proposed method considers chunking of alphabetic letters of an English word into conversion units and considers English and Japanese context information simultaneously to calculate the plausibility of conversion. They have shown experimentally that the proposed method improves the conversion accuracy by 63% compared to a simple method that ignores the plausibility of chunking and contextual information.

**Abduljaleel and Larkey** [54] presented **English to Arabic** statistical transliteration using **grapheme-based** approach. This technique requires no heuristics or linguistic knowledge of either language. This technique learns probabilities between English and Arabic characters from a training sample of pairs of transliterated words from the two languages. Based on these probabilities, it generates Arabic transliterations for unknown English words.

**Yan et al.** [58] described a method for automatically creating and validating candidate Japanese transliterated terms of English words. A phonetic English dictionary and a set of probabilistic mapping rules are used for automatically generating transliteration candidates. A monolingual Japanese corpus is then used for automatically validating the transliterated terms. They have evaluated the usage of the extracted English-Japanese transliteration pairs with Japanese to English retrieval experiments over the CLEF bilingual test collections. The use of automatically derived extension to a bilingual translation dictionary improves average precision, both before and after pseudo-relevance feedback, with gains ranging from 2.5% to 64.8%.

**Virga and Khudanpur** [19] address the problem of transliterating English names using Chinese orthography in support of cross-lingual speech and text processing.
applications. Based on IBM source channel model for statistical transliteration, they demonstrated fully data-driven English to Chinese transliteration system with statistical learning and phoneme-based approach. As compared with Meng et al. [24] the pin-yin error performance of their system (Small MT) is 50.8% whereas the same with the Meng et al. system was 52.5% and with increasing the training size from 2233 to 3625 has further reduced the syllable error rate at (Big MT) 49.1%. They have also evaluated the efficiency of second translation system which maps the pin-yin sequence produced by the previous method to a sequence of Chinese character and obtained character error rate of 12.6%.

Li et al. [11] modeled a joint source channel model for machine transliteration between Chinese and English that allows direct orthographic mapping (DOM) between two different languages. With the n-gram TM model they have automated the orthographic alignment process to derive the aligned transliteration unites from a bilingual dictionary. The bilingual aligning process is integrated into the decoding process in n-gram TM. Unlike other related work where pre-alignment is needed, the new framework greatly reduced the development efforts. The word error rates at 10-best level for both English-Chinese 5.4% (Open), 0.90% (Closed) and Chinese-English 24.6% (Open), 4.8% (Closed) are reported which imply potential error reduction by secondary knowledge source (lookup table). The DOM framework shows a quantum leap in performance with n-gram transliteration model as compare to phoneme-based approach presented by Virga and Khudanpur [19]. The n-gram transliteration model presents an error reduction of 74.6% for Pinyin over the best reported result. Although the framework is implemented on an English-Chinese personal name data set, without loss of generality, it well applies to transliteration of other language pairs such as English/Korean and English/Japanese.

Gao et al. [59, 60] presented a statistical transliteration system between English and Chinese using phoneme-based approach. Unlike traditional rule-based approaches, their method is data-driven. So it is independent of dialect features in Chinese. They modeled the statistical transliteration problem as a direct phonetic symbol transcription model plus a language model for post-adjustment. In addition, it is different from other statistical approaches based on source-channel framework in that they adopted a direct transliteration model, i.e. the direction of probabilistic estimation is consistent with transliteration direction. They selected 200 English names from LDC’s bilingual named entity list, in which 100 of them were seen in the
training set (for close test) and 100 of them were untrained (for open test). Only top-1 machine generated transliteration of each name is chosen for comparison with its standard translation. The evaluation results showed that 54% of the test instances can be transformed to their Chinese translations with an error rate less than 50%. This indicates that majority of their machine generated transliterations tends to be half-correct at least. The close test, where 66% machine generated transcriptions with error rate less than 50%, performs better than open test where 42% (100-58) is with the error rate < 50%. Moreover, the possibility of finding the correct transliteration in top-1 result candidates is fairly low on both tests i.e. only 7% and 10% of the test instances (8.5% in average) end up with the top-1 transliteration that can be considered as correct.

Transliterating a loan word written in a foreign language back to the language of its origin is known as back transliteration. Because of phonetic gaps in one language or the other, the information may be lost about the original word when it is transliterated. Hence, automatically converting transliterated words back into their original form is a real challenge. Bilac and Tanaka [29] modeled a hybrid back transliteration systems for Japanese. They combined the transliterated string segmentation module both phoneme and grapheme-based transliteration models to enhance the back transliteration for Japanese words. The first evaluation set consisted of 150 katakana words extracted from the EDR*, 1995 Japanese corpus and the second test set was from NTCIR-2 test collection. The evaluation of EDR data set results showed a significant improvement by integrating segmentation module into the hybrid system. The top-1 and top-10 transliteration results for EDR test data were 59.33% and 75.33% respectively. Similarly, the top-1 and top-10 transliteration results for NTCIR-2 test data were 62.82% and 83.33% respectively.

In yet another research paper, Bilac and Tanaka [30] proposed similar approach for improving back transliteration by combining information sources. The new combination excluded the earlier segmentation module and a new probabilistic model is used along with both phoneme and grapheme-based transliteration models. They extracted a collection of about 7000 words in katakana together with the corresponding English translation from the EDICT dictionary. About 10% of these

---

entries were left out for evaluation. The top-1 and top-10 best reported results attained were 84.59% and 97.76% respectively.

Later, Bilac and Tanaka [31] extended the statistical, hybrid back transliteration systems to incorporate both Japanese to English and Chinese to English. Rather than producing back transliterations based on grapheme and phoneme model independently and then interpolating the results, they have introduced a method of first combining the sets of allowed rewrites (i.e. edits) and then calculating the back transliterations using the combined set. As expressed in their earlier attempts, the evaluation on both Japanese and Chinese transliterations shows that the direct combination increases robustness and positively affects back transliteration accuracy.

Oh and Choi [61] presented English to Korean transliteration system with statistical learning. Unlike the previous works, they have exploited both grapheme and phoneme information in English-to-Korean transliteration. They have devised three alignment algorithms called EP, PK and EPK alignment, named according to the correspondences dealt with by each algorithm. The EP alignment algorithm finds the most phonetically probable correspondence between English grapheme and phoneme; while the PK alignment algorithm finds the most phonetically probable correspondence between phoneme and Korean grapheme; and the EPK alignment is a hybrid process of EP alignment and PK alignment that deals with correspondences among English grapheme, phoneme and Korean grapheme by combining results of EP and PK alignment using phoneme information as a pivot. The EP and PK alignment algorithm is based on Levenshtein distance (LD) algorithm. Levenshtein distance is a measure of the similarity between two strings. The distance is the number of deletions, insertions, or substitutions required to transform source string into target string. The three machine learning methods described by them are maximum entropy model (MEM), decision tree (DT) and memory-based learning (MBL). In experiments, they found that MBL is the best machine learning method in its task and correct pronunciation is very helpful to generate a correct Korean transliteration. After a comparative evaluation on different datasets with the existing systems, they have achieved 13% ~ 78% performance improvements. [61-63]

The performance of the Named Entity (NE) transliteration system can be improved by considering their information of origin. The practical study of this approach has been carried out by Huang [64]. He proposed Chinese to English cluster-specific named entity transliteration system using grapheme-based approach. He grouped
name origins into a smaller number of clusters, and then trained transliteration and language models for each cluster under a statistical machine translation framework. Given a source NE, he first selected appropriate models by classifying it into the most likely cluster, then the transliteration of the corresponding NE is performed with the matching models. He also proposed a phrase-based name transliteration model, which effectively combines context information for transliteration. The baseline system was a character-based general transliteration model, where 56K NE pairs from all clusters were merged to train a general transliteration model and a language model. Experiments showed substantial improvement on the transliteration accuracy over a state-of-the-art baseline system, significantly reducing the transliteration character error rate from 50.29% to 12.84%.

Karimi et al. [17] proposed consonant-vowel (CV) based algorithms for English-Persian transliteration. Their methods, named CV-model1 and CV-model2, were based on specific patterns of sequences of consonants and vowels in source words. The source and target word-pair (S, T) is first aligned (using giza++) to approximate the correspondence between source and target characters. A consonant-vowel sequence is then built for S by replacing each consonant with C and each sequence of vowels with V. This sequence together with the original characters is then broken into specific patterns such as CVC, CC, VC, and CV. Attaching the corresponding S characters to T characters based on these patterns, transformation rules are generated and the transliteration model is formed. The difference between the two methods, cv-model1 and cv-model2, is that cv-model2 does not keep the original consonant characters in the final model, leaving them as C in the transformation rules that contain vowels. Note that the consonants and vowels were orthographic rather than phonemic. Evaluation of these systems was conducted on an English-Persian transliteration corpus of 16,760 word-pairs, proper names from different origins, created by 26 human transliterators. cv-model1 and cv-model2 resulted in word accuracy of 51.6% and 48.4% (top-1), respectively. A subset of English-only source words, 1,857 pairs, resulted in word accuracy of 61.7% and 60.0% (top-1), respectively, for cv-model1 and cv-model2.

Oh et al. [32] have extended their hybrid English to Korean approach to English to Japanese language pair. Like the previous attempt this system is based on correspondence between graphemes and phonemes i.e. hybrid in nature. Their correspondence based transliteration model has two component functions. The first
function produces the most probable sequence of source phonemes (pronunciation) corresponding to the source grapheme using pronunciation dictionary as well as pronunciation estimation and the role of second function is to produce target graphemes. They have used the same three machine learning algorithms as expressed earlier. Experiments showed that their correspondence based model for English to Japanese is more effective than other transliteration models and its performance was better by about 16-44%.

Additionally, a hybrid model for extracting transliteration equivalents from parallel corpora and improving transliteration performance by using multiple transliteration models are also presented by these authors. [65, 66]

**Zelenko and Aone** [67] have presented grapheme-based transliteration between Arabic, Korean, and Russian to English. They presented two discriminative methods for name transliteration. The methods correspond to local and global modeling approaches in modeling structured output spaces. They did not estimate either direct conditional \( p(e|f) \) or reverse conditional \( p(f|e) \) or even joint \( p(e,f) \) probability models. Both methods do not require alignment of names in different languages and their features are computed directly from the names themselves. They learn automatically a function that directly maps a name \( f \) from one language into a name \( e \) in another language using an existing transliteration dictionary having a set of name pairs like \( \{(f, e)\} \). For language modeling, the list of 100,000 most frequent names downloaded from the US Census website was used. The language model is a 5-gram model with interpolated Good-Turing smoothing. The experimental evaluation of both the methods for name transliteration is compared with the joint probabilistic modeling approach. The highest evaluation accuracy results for Arabic to English with relative distance \( d=1 \) are 31.33 (local), 32.61 (global), 25.75 (probabilistic); Korean to English with relative distance \( d=3 \) are 30.96 (local), 35.28 (global), 26.93 (probabilistic); Russian to English with relative distance \( d=1 \) are 44.62 (local), 46.28 (global), 39.67 (probabilistic). According to accuracy results the discriminative methods outperform probabilistic modeling, with the global discriminative modeling approach achieving the best performance in all the three languages.

**Sherif and Kondrak** [68] presented a grapheme-based bootstrapping approach to train a memory-less stochastic transducer for the task of extracting transliterations from an **English-Arabic** bitext. The bootstrapped transducer learns its similarity
metric as proposed by Ristad and Yianilos [69] from the data in the bitext, and thus can function directly on strings written in different writing scripts without any additional language knowledge. The transducer is able to learn, edit distance costs between disjoint sets of characters representing different writing scripts without any language-specific knowledge. Thus, the proposed bootstrapped stochastic transducer is completely language-independent. The transducer approach, however, requires a large set of training examples which is a limitation not present in the fuzzy matching algorithm as proposed by Freeman et al. [70]. The first 1000 documents in the parallel news data were used for testing. The results of the NER detection task showed that both the transducer and the fuzzy matching algorithm have shown the accuracy figure of 74.6% in their performance. On the other hand they outperformed when compared with the Levenshtein edit distance [62] algorithm and ALINE algorithm [71] having accuracy of 69.3% and 71.9% respectively as compared by the authors.

Yoon et al. [72] use phonetic distinctive features and phonology-based pseudo features to learn both language-specific and language universal transliteration characteristics. The distinctive features are the characteristics that define the set of phonemic segments (consonants, vowels) in a given language. Pseudo features capture sound change patterns that involve the position in the syllable. Distinctive features and pseudo features are extracted from source and target language training data to train a linear classifier. The classifier computes compatibility scores between English source words and target-language words. When several target-language strings are transliteration candidates for a source word, the one with the highest score is selected as the best transliteration. The method was evaluated using parallel corpora of English with each of four target languages. Named entities were extracted from the English side and were compared with all the words in the target language to find proper transliterations. The base line presented for the case of transliteration from English to Arabic achieves Mean Reciprocal Rank (MRR) of 0.66 and this method improves its results by 7% where as no gain is found in Chinese (0.74). This technique involves knowledge about phonological characteristics of consonants based on their position in the word, which requires expert knowledge of the language. In addition, conversion of terms into a phonemic representation poses hurdles in representing short vowels in Arabic and will have similar behavior in Hebrew. Moreover, English to Arabic transliteration is easier than Arabic to English,
because in the former, vowels should be deleted whereas in the latter like Shahmukhi script they should be generated. The base line MRR presented for English to Hindi and Korean pairs are 0.87 and 0.82 while the method considering all features improves its results by showing MRR 0.91 and 0.84 respectively.

In his master thesis David Matthews [73] gave results using Moses Phrase-based Statistical Machine Translation to conduct experiments of transliteration of proper names between both English-Chinese and Arabic-English. They have designed separate transliteration and target side language models and combined them during decoding to find the most likely transliteration. The parallel corpus from which the translation model is acquired contains approximately 2500 pairs, which are part of a bilingual person names corpus (LDC-2005-G02). This biases the model towards transliterating person names. Through experiments they find optimal parameter settings and investigate the effects of varying the size of both language and transliteration models. Moses outperforms the baseline model in both directions and in both language pairs. The best forward transliteration accuracy achieved from English to Chinese was 37.8%, the best back transliteration accuracy from Chinese to English was 34.8%. The best forward transliteration accuracy achieved from Arabic to English was 43.0%, the back transliteration accuracy from English to Arabic was 39.2%. The Moses’ performance is comparable to recent work in both English-Chinese [74] and Arabic-English [75].

Karimi et al. [76] investigated both Persian-English and English-Persian transliteration by improving their consonant-vowel based methods [17] in two ways: a new alignment algorithm that replaced GIZA++, and a new method of forming consonant-vowel sequences. The new alignment algorithm was based on collapsed consonant-vowel sequences of both source and target words, and the frequency of aligning their substrings. These homogeneous sequences were broken into their constituent substrings based on the frequency of alignment in a training corpus. A similar concept of grouped consonants and vowels was proposed for the transliteration stage where transformation rules that were consistent with the alignment step were formed by using the boundaries of consonants and vowels. This method was named cv-model3. Evaluations on a corpus of 16,670 English-Persian word-pairs from different origins showed a word accuracy of 55.3%, and for a sub-collection of all English source names it was 67.4%. Persian-English transliteration on a collection of 2,010 pairs led to 39.0% (top-1) word accuracy.
Oh and Isahara [77] presented a combined approach for English-Korean and English-Japanese transliteration using support vector machines (SVM) along with maximum entropy models (MEM) to re-rank the outputs of individual transliteration systems. These individual systems were drawn from a variety of phoneme-based, phonetic-based, and even hybrid methods. Both machine learning components, SVM and MEM, were trained using confidence score, language model, and Web frequency features. The Web frequency parameter was adapted from other Web based systems, similar to the method proposed by Al-Onaizan and Knight [78] which counts the co-occurrence frequencies of the transliteration pair on the Web. For evaluation corpora contained named entries of 7,172 pairs of English-Korean and 10,417 pairs of English-Japanese from the EDICT dictionary. Using seven individual systems, they reported 87.4% (top-1) word accuracy for English-Japanese transliteration, and 87.5% for English-Korean, when the MEM-based approach was used. For the SVM-based approach these results were 87.8% and 88.2%, respectively.

Goldwasser and Roth [79] suggested a discriminative method for identifying named entity (NE) transliteration pairs in English-Hebrew. Given a word pair $(ws,wt)$, where $ws$ is an English NE, the system determines whether $wt$, a string in Hebrew, is its transliteration or not. The classification is based on pair wise features: sets of substrings are extracted from each of the words, and substrings from the two sets are then coupled to form the features. The accuracy of correctly identifying transliteration pairs in top-1 and top-5 was 52% and 88%, respectively.

Goldberg and Elhadad [80] put forward a loosely supervised method for non-contextual identification of transliterated foreign words in Hebrew texts. The method is a Naive-Bayes classifier which learns from noisy data. Such data are acquired by over generation of transliterations for a set of words in a foreign script using mappings from the phonemic representation of words to the Hebrew script. Precision and recall obtained are 80% and 82% respectively.

Hermjakob et al. [81] outlined a method for identifying Name Entities that should be transliterated in Arabic texts. The method first tries to find a matching English word for each Arabic word in a parallel corpus, and tag the Arabic words as either names or non-names based on a matching algorithm. This algorithm uses a scoring model which assigns handcrafted costs to pairs of Arabic and English substrings, allowing for context restrictions. A number of language specific heuristics, such as those considering only capitalized words as candidates and using lists of ending
words, are used to enhance the algorithm’s accuracy. The tagged Arabic corpus is then divided: One part is used to collect statistics about the distribution of name/non-name patterns among tokens, bigrams and trigrams. The rest of the tagged corpus is used for training. The precision of the identification task is 92.1% and its recall is 95.9%. This work also presents a transliteration model, which is integrated into a machine translation system. Its accuracy as measured by the percentage of correctly translated names is 89.7%.

Kirschenbaum and Wintner [82] worked on a Hebrew to English transliteration method in the context of a machine translation system. For the purpose of identification of Hebrew Terms To-be Transliterated (TTT), rather than translated, they proposed a method by using machine learning from a semi-automatically acquired training corpus. The proposed classifier reduced more that 38% of error made by a naïve baseline system. Similar to our research problem, a major difficulty faced by them resulted from the fact that in the Hebrew orthography (like Arabic) words are represented as sequences of consonants whereas vowels are only partially and very inconsistently represented. Even letters that are considered as representing vowels may sometimes represent consonants. As a result, the mapping between Hebrew orthography and phonology is highly ambiguous. The SMT based transliteration model is trained with a parallel corpus extracted from Wikipedia. The system’s overall accuracy is about 76% for top-1 results and 92% for top-5 results.

Karimi [83] constructed a black-box framework for both English-Persian and Persian-English transliteration. Multiple phoneme-based transliteration systems were aggregated into one system with the combination method being a mixture of a Naïve-Bayes classifier and a majority voting scheme. The English-Persian system was trained and tested on a controlled corpus of 1,500 English words transliterated by seven human transliterators. Similarly, the Persian-English system was evaluated on a corpus of 2,010 Persian person names accompanied by variants of their possible English transliterations. Empirically, a significant performance improvement was reported in both directions. They found 85.5% word accuracy for English-Persian and 69.5% word accuracy for the reverse combined system.

Instead of using conventional NLP techniques, Deselaers et al. [84] demonstrated that deep belief networks (DBNs) have certain properties which are very interesting to exploit for transliteration and that a combination with conventional techniques leads to an improvement over both components on an Arabic-English transliteration
task. One advantage that DBN model has is in principle is that the DBN approach is fully bi-directional. Their proposed model is similar to Collobert and Weston [85] approach that used the same machine learning techniques but differed in encoding and the processing. First, the system learns from two independent generative models, one for the source input and one for the target output. Then, these two models are combined into a source-to-target encoding/decoding system. Deep belief networks consisted of multiple layers of restricted Boltzmann machines (RBMs). RBMs are stochastic generative artificial neural networks with restricted connectivity. DBNs are built from RBMs by first training an RBM on the input data. Then a second RBM is built on the output of the first one and so on until a sufficiently deep architecture is created. A corpus of 10,084 personal names in Arabic and their transliterated English ASCII representation (LDC corpus LDC2005G02) is used. The Arabic names are written in the usual way, i.e. lacking vowels and diacritics. 1,000 names were randomly sampled for system development and evaluation. They evaluated character error rate (CER) in place of commonly used word error rate (WER). The baseline performance of the proposed system rolled up to 24% CER on the development data and 22.7% CER on the evaluation data.

Wutiwiwatchai and Thangthai [25] presented a bidirectional Thai-English machine transliteration applied on the NEWS 2010 transliteration corpus. The proposed model is similar to Jiang et al. [26], which introduced transliteration among Chinese and English names based on syllable units and determined the best candidate using the statistical n-gram model. The system relies on segmenting source language words into syllable-like units, finding unit’s pronunciations, consulting a syllable transliteration table to form target-language word hypotheses, and ranking the hypotheses by using syllable n-gram. The training set composes of 24,501 word pairs and two test sets comprising 2,000 words from English-to-Thai and 1,994 words from Thai-to-English are used for evaluation. The approach yielded 84.2% and 70.4% mean F-scores on English-to-Thai and Thai-to-English transliteration.

The system presented by Finch and Sumita [86] is a combination of SMT and Joint Multigram technique for direct transliteration. This technique makes no language specific assumptions, uses no dictionaries or explicit phonetic information; the process transforms sequences of tokens in the source language directly into sequences of tokens in the target. All the language pairs in the experiments were transliterated by applying a single unified technique. It is a hypothesis rescoring
approach that integrates the models of two state-of-the-art techniques: phrase-based statistical machine translation (SMT) and a joint multigram model. The joint multigram model was used to generate an n-best list of transliteration hypotheses that were re-scored using the models of the phrase-based SMT system. Both models’ scores for each hypothesis were linearly interpolated to produce a final hypothesis score that was used to re-rank the hypotheses. In the experiments on development data, the combined system was able to outperform both of its component systems substantially. The joint multigram system outperformed the phrase based SMT system. The top-1 accuracy achieved on language pairs namely English-Thai, Thai-English, English-Hindi, English-Tamil, English-Kannada, English-Japanese, Arabic-English and English-Bangla was 0.412, 0.397, 0.445, 0.390, 0.371, 0.378, 0.403 and 0.412 respectively. However, in particular, their proposed methodology requires a significant improvement in Arabic-English language pair when compared with top ranked system. In experiments run on the English-to-Japanese katakana task, the joint multigram system in isolation achieved an F-score of 0.837 on development data, whereas the SMT system in isolation achieved an F-score of 0.824. When integrated the models of the systems complemented each other well, and on the same English-Japanese task the integrated system achieved an F-score of 0.843.

2.2.2 Machine Transliteration Systems for Indian Languages

According to censes of India in 1991 and 2001, there were 18 and 22 scheduled languages’ respectively in India. The 2001 census recorded 29 individual languages as having more than 1 million native speakers (0.1% of total population). Taking into account major languages in the country, machine aided transliteration from one language to another would yield an overwhelmingly large number of pairs and addressing all such pairs for transliteration is really a challenging task for developing a common solution platform.

The languages of India are phonetic in nature and hence the writing system for any Indic language maps the sounds of the aksharas to specific shapes. The basic set of aksharas for most languages consists of sixteen vowels and about forty consonants. Whereas, the actual rules for forming consonant vowel (CV) combinations and conjunct characters vary from script to script. During the past several years, different

\[ \text{http://en.wikipedia.org/wiki/Official_languages_of_India} \]
methods have been introduced to prepare Indian language documents by entering the
text through specific transliteration schemes. Except the ITRANS (Indian language
Transliteration package) most of the other transliteration tools for Indian languages are
MADURAI, ADAMI and ADHAWIN for Tamil; GIST-SHELL, iLEAPE, GIST-
UTRANS, GIST-Mobile, XLIT, SANSCRIPT and Google IME for Hindi and other
Indian languages; Rice Inverse Transliteration (RIT) system for Telugu; GTrans for
Gurmukhi script of Punjabi language. [87-95]

<table>
<thead>
<tr>
<th>SN</th>
<th>Language</th>
<th>SN</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Assamese</td>
<td>10</td>
<td>Marathi</td>
</tr>
<tr>
<td>2</td>
<td>Bengali</td>
<td>11</td>
<td>Nepali</td>
</tr>
<tr>
<td>3</td>
<td>Gujarati</td>
<td>12</td>
<td>Oriya</td>
</tr>
<tr>
<td>4</td>
<td>Hindi</td>
<td>13</td>
<td>Punjabi</td>
</tr>
<tr>
<td>5</td>
<td>Kannada</td>
<td>14</td>
<td>Sanskrit</td>
</tr>
<tr>
<td>6</td>
<td>Kashmiri</td>
<td>15</td>
<td>Sindhi</td>
</tr>
<tr>
<td>7</td>
<td>Konkani</td>
<td>16</td>
<td>Tamil</td>
</tr>
<tr>
<td>8</td>
<td>Malayalam</td>
<td>17</td>
<td>Telugu</td>
</tr>
<tr>
<td>9</td>
<td>Manipuri</td>
<td>18</td>
<td>Urdu</td>
</tr>
</tbody>
</table>

The published research papers modeling various transliteration systems using Indian
language are discussed here.

A modified joint source–channel model along with a number of alternatives has been
proposed by Ekbal et al. [20]. A Bengali-English machine transliteration system has
been developed based on the proposed model. They developed a framework that
allows direct orthographical mapping between two languages that are of different
origins employing different alphabet sets. In proposed model, aligned transliteration
units along with their context are automatically derived from a bilingual training
corpus to generate the collocation statistics. The transliteration units in Bengali
words take the pattern C+M where C represents a vowel or a consonant or a conjunct
and M represents the vowel modifier or matra. The English transliteration units are
of the form C*V* where C represents a consonant and V represents a vowel. The
system has been trained to transliterate person names from Bengali to English. It uses
the linguistic knowledge of possible conjuncts and diphthongs in Bengali and their
equivalents in English. The system has been evaluated and it has been observed that
the modified joint source-channel model performs best with a word agreement ratio
of 69.3% and a transliteration unit agreement ratio of 89.8%.
UzZaman et al. [96] presented a direct phonetic mapping based transliteration scheme for Roman English to Bangla which produces intermediate code strings that facilitate matching pronunciations of input and the desired outputs. The traditional direct mapping schemes generally show one-to-one mapping but alternatively, they have proposed a partially many-to-one mapping scheme. They also provide for a more conventional direct phonetic mapping in special circumstances i.e., direct mapping technique using table-driven approach between the English and Bangla alphabet, and the second form is phonetic lexicon–enabled mapping. System testing is performed on an input of approximately 2500 words from Bangla newspaper articles after converting them into Roman (English) format. The evaluation of developed prototype shows that there are 32% words which are not found in the lexicon but out of them 23% are handled properly by direct mapping. The remaining unhandled words have 7% inflected words and 2% errors cases.

Vijaya et al. [97] proposed a method where transliteration problem between English to Tamil has been modeled as a sequence labeling problem using Memory-based learning. They have reformulated the transliteration problem as a sequence labeling problem from one language alphabet to another. This process has two levels of decoding: segmentation of the source string into transliteration units and relating the source language transliteration units with units in the target language by resolving different combinations of alignments and unit mappings. They have expressed IB1-IG algorithm for the memory-based learning which is used as a form of supervised learning based on similarity-based reasoning and contextual information is used to train the model. To achieve system efficiency IGTree decision tree technique as proposed by Daelemans and Zavrel [98] is implemented which combines two algorithms: one for compressing a case base into a tree, and one for retrieving classification information from these trees. A parallel corpus of 30,000 person names and 30,000 place names and used in the training phase. Finally, the complete transliteration model has three data structures; a lexicon, a case base for known source language alphabets and a case base for unknown alphabets automatically extracted from the given aligned corpus and indexed using IGTree. The system evaluation is performed with 1000 English names that were not present in the training corpus. The top-1 and top-5 accuracy results of the proposed English to Tamil system are 84.16% and 93.33% respectively.
Surana and Singh [99] observed empirically that the information about word origin is an important factor in achieving higher accuracy in transliteration. They proposed a multilingual transliteration mechanism for Hindi and Telugu Indian languages using different techniques based on the word origin. This method first identifies the class (foreign or Indian) of the word on the source side. Based on the class, one of the two methods is used for transliteration. Easily creatable mapping tables and a fuzzy string matching algorithm are then used to get a ranked list of target words. To evaluate the method a list of 200 words containing both Indian and foreign words is created. The best MRR results presented for English to Hindi and Telugu pairs are 0.87 and 0.82 with respective precision of 80% and 71%. As already discussed Yoon et al. [72] have also reported MRR score for four languages including Hindi as 0.91 and 0.89 under different test conditions but they have tested their algorithm only for named entities.

Malik et al. [100] proposed a Hindi Urdu machine transliteration (HUMT) system using finite-state transducers (FSTs). The Hindi-Urdu transliteration is performed using universal intermediate transcription (UIT) that is a proposed encoding scheme to uniquely encode natural languages into ASCII. The UIT encoding is for the same pair on the basis of their common phonetic repository in such a way that it can be extended to other languages like Arabic, Chinese, English, French, etc. In addition, as claimed by the authors, the same transducer that encodes a Hindi or Urdu text into UIT can be used in the reverse direction to generate Hindi or Urdu text from the UIT encoded text. For evaluation, a Hindi corpus containing 374,150 words and an Urdu corpus with 38,099 words was developed. The Hindi corpus is extracted from the Hindi WordNet developed by the Resource Center for Indian Language Technology Solutions, IIT Bombay, India, whereas the Urdu corpus was developed manually from a book. The HUMT system has shown 97.50% accuracy when it was applied on the Hindi-Urdu corpora containing 412,249 words in total.

Ganesh et al. [101] presented a discriminative model having simple statistical technique for transliteration using HMM alignment and Conditional Random Fields (CRFs). They have explored this technique for Hindi-English CLIR transliteration task. The approach for transliteration is divided into two phases. The first phase induces character alignments over a word-aligned bilingual corpus, and the second phase uses statistics over the alignments to transliterate the source language word and generate the desired number of target language words. The first HMM model was
developed using HMM alignment and conditional probabilities derived from counting the alignments whereas the second HMM & CRF model was developed using HMM alignment and CRF for generating top-n transliterations. The evaluation of both HMM and HMM & CRF models with 30,000 in-corpus words has shown top-5 accuracy as 74.2% and 76.5% respectively. Another transliteration evaluation based on 1,000 out of corpus words has shown top-5 accuracy as 69.3% and 72.1%. Similarly, to perform CLIR based testing, both the models were evaluated with CLEF 2007 documents and 50 topics. The overall evaluation results demonstrated that HMM & CRF model works better than using HMMs alone. The Mean Average Precision (MAP) value for both HMM and HMM & CRF model was found to be 0.1347 and 0.1499 respectively.

Chinnakotla et al. [102] presented Hindi to English and Marathi to English CLIR systems developed as an ad-hoc Bilingual task. This is a query translation based approach using bi-lingual dictionaries. Those query words which are not found in the dictionary are transliterated using a simple lookup table based transliteration approach. The resultant transliteration is then compared with the index items of the corpus to return the "n" closest English index words of the given Hindi or Marathi word. The resulting multiple transliteration choices for each query word are disambiguated using an iterative page-rank style algorithm, which makes use of term-term co-occurrence statistics to produce the final translated query. Using the above approach for Hindi, the Mean Average Precision (MAP) of 0.2366 (61.36%) was achieved after using title and a MAP of 0.2952 (67.06%) was achieved using title and description. For Marathi, they have achieved a MAP of 0.2163 (56.09%) using title.

Lehal [103] presented a highly accurate transliteration system between Gurmukhi to Shahmukhi scripts. A direct method approach for character mapping was used and the system learning approach was rule-based. The transliteration system has used Gurmukhi spellchecker, Gurmukhi-Shahmukhi dictionary and Shahmukhi corpus. The paper discusses the important issues in Gurmukhi to Shahmukhi transliteration with statistical results. This is a Unicode based system which can transliterate any Gurmukhi text into Shahmukhi at more that 98.6% accuracy at word level.

Haque et al. [104] presented Phrase-based statistical English-Hindi transliteration in the NEWS 2009 Machine Transliteration Shared Task adding source context modeling into log-linear phrase-based statistical machine translation. The main
advantage is that source context features enables the exploitation of source similarity in addition to target similarity, as they have modeled by the language model. They used a memory-based classification framework that further enables estimation of these features while avoiding data sparseness problems. For the standard submission 10,000 Name Entities from the NEWS 2009, English-Hindi training data are used and for the non-standard submissions, the additional English-Hindi parallel person names data of 105,905 distinct name pairs are taken from the Election Commission of India. They carried out experiments both at character and transliteration unit (TU) level. The accuracy of the baseline system is found to be 0.391 at the TU level that is much higher as compared to the Character level i.e. 0.290 on standard dataset. However, the highest accuracy that was achieved using the larger dataset with the TU level is 44.5%.

Vardarajan and Rao [105] described a method for transliterating an English string to a foreign language string and evaluated it on five different languages, including Tamil, Hindi, Russian, Chinese, and Kannada. The method involves deriving substring alignments from the training data and learning a weighted finite state transducer from these alignments. The defined $\varepsilon$-extension Hidden Markov Model is used to derive alignments between training pairs and additional heuristic to extract the substring alignments. The method involves only two tunable parameters that can be optimized on held-out data. The system evaluation is performed with the standard track data provided by the NEWS 2009 shared task on different languages. The top-1 accuracy results for Tamil, Hindi, and Kannada are 0.327, 0.398, and 0.235 respectively. However, the F-score results for Tamil, Hindi, and Kannada are 0.870, 0.855, and 0.817 respectively. [105]

Das et al. [106] presented their English to Hindi named entities transliteration in the NEWS 2009 Machine Transliteration Shared Task. They have combined three transliteration models that can generate the Hindi transliteration from an English named entity (NE). An English NE is divided into Transliteration Units (TUs) with patterns C*V*, where C represents a consonant and V represents a vowel. Similarly, the Hindi NE is divided into TUs with patterns C+M?, where C represents a consonant or a vowel or a conjunct and M represents the vowel modifier or matra. The TUs are the lexical units for machine transliteration. The system considers the English and Hindi contextual information in the form of collocated TUs simultaneously to calculate the plausibility of transliteration from each English TU to
various Hindi candidate TUs and chooses the one with maximum probability. The system learns the mappings automatically from the bilingual NEWS training set being guided by linguistic features/knowledge. The system considers the linguistic knowledge in the form of conjuncts and/or diphthongs in English and their possible transliteration in Hindi. The output of the mapping process is a decision-list classifier with collocated TUs in the source language and their equivalent TUs in collocation in the target language along with the probability of each decision obtained from the training set. Linguistic knowledge is used in order to make the number of TUs in both the source and target sides equal. A direct example base is maintained that contains the bilingual training examples that do not result in the equal number of TUs in both the source and target sides during alignment. The Direct example base is checked first during machine transliteration of the input English word. If no match is obtained, the system uses direct orthographic mapping by identifying the equivalent Hindi TU for each English TU in the input and then placing the Hindi TUs in order. The three adopted transliteration models includes a joint source-channel model [11], a trigram model where the previous and the next source TUs are considered as the context, and an improved modified joint source-channel model [20]. However, the output of the modified joint source-channel model is given more priority during output ranking followed by the trigram and the joint source-channel model. Upon evaluation for standard run, the system demonstrated accuracy (top-1) of 0.471 was found and the mean F-Score stood at 0.861. On the other hand, the two non-standard runs yielded the accuracy (top-1) and mean F-scores of 0.389 and 0.831 respectively in the first one and 0.384 and 0.828 respectively in the second one. The non-standard runs resulted in substantially worse performance than the standard run. The reason for this is the ranking algorithm they used for the output and the types of tokens itself present in the test set.

Chinnakotla and Damani [107] presented a phrase-based statistical machine translation approach for transliteration between English-Hindi, English-Tamil and English-Kannada language pairs at NEWS 2009. They modeled phrase-based Statistical Machine Translation (SMT) approach to transliteration where the words are replaced by characters and sentences by words. They employed the standard SMT tools like GIZA++ for learning alignments and Moses for learning the phrase tables and decoding. Increased transliteration accuracy was achieved on tuning the Character Sequence Model (CSM) related parameters like order of the CSM, weight
assigned to CSM during decoding and corpus used for CSM estimation, besides tuning the standard SMT parameters. The NEWS 2009 Test Set Results show that the top-1 accuracy of English-Hindi Standard run is 0.42 and Non-standard is 0.49. Similarly, top-1 accuracy of English-Tamil Standard run is 0.41 and English-Kannada Standard run is found to be 0.36.

Bushra and Tafseer [108] performed a detailed analysis of existing work on Hindi and Urdu transliteration systems and reported the unresolved issues that are still not addressed in the existing systems. The main systems they focused on were Hindi to Urdu transliteration performed by CRULP [109] and Malik [110]. After reviewing the output of existing Hindi to Urdu transliteration system they found that there are issues that are still needed to be solved. Generally, the issues in Hindi to Urdu conversion such as similar sounding Urdu characters, Hindi characters with similar shape and Izafat form of Urdu, are beyond the scope of character based mappings. Authors have presented the theoretical solutions to many of those issues that go beyond simple transliteration. They suggested post processing of the transliterated output and they resolved issues caused by writing convention differences by consulting specific word lists or lexical resources.

The Hindi to English, English to Hindi, and Persian to English transliteration tasks proposed by Chinnakotla et al. [1] are based on Character Sequence Modeling (CSM) on the source side for word origin identification. Thereafter, for these resource scarce languages, a manually generated non-probabilistic rule-based character mapping is used for generating transliteration candidates, and then CSM is used again on the target side for ranking the generated candidates. The above approach has shown that by using just the monolingual resources in conjunction with manually created rule base, one can achieve reasonable transliteration performance when compared to that of baseline statistical systems trained using parallel corpora. For training the CSM in Hindi, a Hindi web crawl of 427,067 documents was obtained from an Indian Search Engine company known as Guruji. For evaluation of English to Hindi transliteration, they used standard NEWS 2009 shared task dataset. The evaluation results of Hindi to English development set as well as test set results on NEWS 2009 dataset are 70.1% (development) and 69.8% (test). Similarly, the evaluation results of reverse English to Hindi transliteration are 60.7% (development) and 59.2% (test). The Persian to English rule-based system was obtained from Karimi [83] and it has been augmented slightly with the help of English to IPA
Lehal et al. [111] presented a high accurate Hindi to Urdu transliteration system. They have used grapheme-based approach for character alignment between the two scripts. The transliteration generation approach is rule-based as well as statistical in nature. They have combined different approaches and modeled a trigram language model to overcome the shortcomings of the existing Hindi to Urdu transliteration systems and presented a system which can transliterate any Hindi Unicode text to Urdu at 97.12% accuracy at word level. The transliteration system used Hindi spellchecker, Hindi-Urdu dictionary and Hindi, Urdu monolingual corpora for the task.

Antony et al. [112] addressed the problem of transliterating English to Kannada language using SVM kernel. The transliteration scheme is modeled using sequence labeling method. The framework based on data driven method and one to one mapping approach simplify the development procedure of transliteration system and facilitates better improvement in transliteration accuracy when compared with that of other state-of-the-art machine learning algorithms. The model is trained on 40,000 words containing Indian place names. The system testing is performed with 1000 place names that were out of corpus and the evaluation results showed that the top-1 accuracy of the system is 81.25% and top-5 accuracy is 91.32%.

Saravanan et al. [113] carried out two cross lingual evaluation tasks, namely the Hindi-English and Tamil-English. Their core CLIR engine employed a language modeling based approach using query likelihood based document ranking and a probabilistic translation lexicon learned from English-Hindi and English-Tamil parallel corpora. They employed two specific techniques to deal with out-of-vocabulary terms in the cross lingual runs i.e., First, generating transliterations directly or transitively, and second, mining possible transliteration equivalents from the documents retrieved in the first-pass. They have experimentally shown that each of these techniques significantly improved the overall retrieval performance of cross lingual IR system. In cross lingual tasks, the systems achieved a peak performance of a Mean Average Precision (MAP) of 0.4977 in Hindi-English and 0.4145 in the Tamil-English. The post-task analyses indicate that the mining of appropriate

(International Phonetic Alphabet) and Persian to IPA mappings. The Persian to English development set accuracy results were 49.7% and this performance is comparable to the CRF (baseline) system’s result of 48.3% while the SMT (baseline) system does much better at 67%.

Lehal et al. [111] presented a high accurate Hindi to Urdu transliteration system. They have used grapheme-based approach for character alignment between the two scripts. The transliteration generation approach is rule-based as well as statistical in nature. They have combined different approaches and modeled a trigram language model to overcome the shortcomings of the existing Hindi to Urdu transliteration systems and presented a system which can transliterate any Hindi Unicode text to Urdu at 97.12% accuracy at word level. The transliteration system used Hindi spellchecker, Hindi-Urdu dictionary and Hindi, Urdu monolingual corpora for the task.

Antony et al. [112] addressed the problem of transliterating English to Kannada language using SVM kernel. The transliteration scheme is modeled using sequence labeling method. The framework based on data driven method and one to one mapping approach simplify the development procedure of transliteration system and facilitates better improvement in transliteration accuracy when compared with that of other state-of-the-art machine learning algorithms. The model is trained on 40,000 words containing Indian place names. The system testing is performed with 1000 place names that were out of corpus and the evaluation results showed that the top-1 accuracy of the system is 81.25% and top-5 accuracy is 91.32%.

Saravanan et al. [113] carried out two cross lingual evaluation tasks, namely the Hindi-English and Tamil-English. Their core CLIR engine employed a language modeling based approach using query likelihood based document ranking and a probabilistic translation lexicon learned from English-Hindi and English-Tamil parallel corpora. They employed two specific techniques to deal with out-of-vocabulary terms in the cross lingual runs i.e., First, generating transliterations directly or transitively, and second, mining possible transliteration equivalents from the documents retrieved in the first-pass. They have experimentally shown that each of these techniques significantly improved the overall retrieval performance of cross lingual IR system. In cross lingual tasks, the systems achieved a peak performance of a Mean Average Precision (MAP) of 0.4977 in Hindi-English and 0.4145 in the Tamil-English. The post-task analyses indicate that the mining of appropriate
transliterations from the top results of the first-pass retrieval achieved enhanced the cross lingual performance of the system overall, in addition to enhancing individual performance of more queries.

**Khapra et al.** [114] employed the methodology of bridge transliteration. They addressed the salient question that, is it possible to develop a practical machine transliteration system between X and Y, by composing two intermediate X $\rightarrow$ Z and Z $\rightarrow$ Y transliteration systems. Most state of the art approaches for machine transliteration are data driven and require massive parallel names corpora between languages. They explored the ways of reducing high resource requirement by leveraging the available parallel data between subsets of n languages transitively. They tested empirically that a reasonable quality transliteration engines may be developed between two languages, X and Y, even when no direct parallel names data exists between them, but only transitively through language Z. In addition, they have also shown that the performance of such transitive transliteration systems is on par with direct transliteration systems in practical applications such as CLIR systems. The performance has been evaluated with four different transitional systems between English, Indic, Semitic and Slavic languages. In each case, they observed that the transitional systems have a slightly lower quality, with an absolute drop in accuracy (ACC-1) of less than 0.05 (relative drop under 10%), and an absolute drop in Mean F-Score of 0.02 (relative drop under 3%).

**Kumaran et al.** [115] extended the idea of compositional transliteration systems to where multiple transliteration components were composed either to provide new transliteration functionality or to enhance the existing transliteration quality between a given pair of languages. The two distinct configurations "serial" and "parallel" have been proposed for compositional systems. The serial compositional transliteration systems chained individual transliteration components in a serial manner to enable creation of transliteration functionality for a given pair of languages with no parallel names corpora between them. The parallel compositional transliteration system aggregated the transliteration evidence from multiple transliteration paths to improve the quality of a given transliteration system. In addition, they also formulated a measure WAVE (n-gram) to measure the transliterability between a given ordered language pair. To validate the utility of the compositional systems, a set of experiments was conducted among English and three Indian languages, namely,
Hindi, Marathi and Kannada. Experiments showed that quality-wise the serial compositional systems do not degrade drastically as compared to baseline direct transliteration systems. In general, the relative drop in accuracy of appropriately designed compositional systems is found to be <10% of that of the corresponding direct systems. Similarly, there is an improvement of up to 8% in transliteration accuracy achieved by parallel methodology over the direct transliteration systems.

2.2.3 Review of Arabic Script Diacritization

Modern written texts in Arabic, Hebrew and other derived languages such as Urdu, Persian, Punjabi (Shahmukhi) etc are composed in script that leaves out most of the vowels of the words. Such languages pose a considerable ambiguity the word level to Natural Language Processing [116]. This is because many words that have different vowel patterns may appear identical in a vowel-less setting. In Hebrew, Levinger et al. [117] computed that 55% out of 40,000 word tokens taken from a corpus of the Israeli daily Ha’aretz were ambiguous. In Arabic, there are almost five possible morphological analyses per word on average [118]. In most of such languages, vowels are marked by both letters and diacritics. In Hebrew, there are twelve different vowel diacritics, and in general, most diacritics are left out of modern script. In Arabic, there are six vowels, which can be divided into three pairs consisting of a short vowel and a long vowel. Each pair corresponds to a different phonetic value. In written Arabic text, the short vowels are generally left out. Surprisingly, native speakers of Arabic or Hebrew can, in most cases, accurately vocalize words in text based on their context and the speaker’s knowledge of the grammar and lexicon of the language. However, speakers of Hebrew are not as successful in restoring the exact vowel diacritics of words. The vowel restoration in such languages is a non-trivial task.

Diacritic restoration has been receiving increasing attention and has been the focus of several studies. The researchers have developed rule-based, semi-automatic and fully automatic diacritization methods in the literature. Even though the methods proposed for diacritic restoration have been maturing and improving over time, they are still limited in terms of coverage and accuracy as discussed below.

El-Sadany and Hashish [119] proposed a rule-based method that uses morphological analyzer for vowelization. The method is semi-automatic vowelization of Arabic verbs. Another rule-based grapheme to sound conversion
approach was developed by El-Imam [120]. The main drawbacks of these rule-based methods were that it is difficult to maintain the rules up-to-date and extend them to other Arabic dialects. Also, new rules are required due to the changing nature of any “living” language.

Gal [116] demonstrated the use of a statistics based approach for vowel restoration in both Arabic and Hebrew languages. A fully automatic approach to diacritization was presented. An HMM based bigram model was used for decoding diacritized sentences from non-diacritized sentences. The technique was applied to the Quran and the word error was 14% (incorrectly diacritized words). This method is a white-space delimited word based approach that restores only vowels; a subset of all diacritics. This has shown that HMMs are a useful tool for computational processing of Semitic languages, and these models can be generalized to other languages too. For the task of vocalizing the vowels according to their phonetic classification, the proposed system achieves an accuracy of 87% for Hebrew and for the task of restoring the exact vowel pattern the reported accuracy of 81% for Hebrew texts and 86% for Arabic texts was achieved.

A first attempt at developing an automatic diacritizer for dialectal speech was made by Kirchhoff et al. [121]. The basic approach was to use a small set of parallel script and diacritized data (obtained from the ECA CallHome corpus) and to derive diacritization rules in an example-based way. This entirely knowledge free approach achieved a 16.6% word error rate.

In the same area of automatic speech recognition, Vergyri and Kirchhoff [122] presented a method to restore diacritics in conversational Arabic by combining morphological and contextual information with an acoustic signal. Diacritization is treated as an unsupervised tagging problem where each word is tagged as one of the many possible forms provided by the Buckwalter’s [55] morphological analyzer. The Expectation Maximization (EM) algorithm is used to learn the tag sequences.

In a technical report at IBM, Emam and Fisher [123] proposed an example based hierarchical top-down approach for the diacritization problem. First, the training data is searched hierarchically for a matching sentence. If there is a matching sentence, the whole utterance is used. Otherwise the search is for matching phrases, then words to restore diacritics. If there is no match at all, character n-gram models are used to diacritize each word in the utterance.
A weighted finite state machine based algorithm is proposed by Nelken and Shieber [34] to restore diacritics. This method employs characters and larger morphological units in addition to words. In their algorithm, a character based generative diacritization scheme is enabled only for words that do not occur in the training data. The model transduces fully diacritized Arabic text weighted according to a language model into undiacritized text. To restore diacritics they used Viterbi decoding, a standard algorithm for efficiently computing the best path through an automation, to reconstruct the maximum likelihood diacritized word sequence that would generate a given undiacritized word sequence. The model is constructed as a composition of several weighted finite-state transducers (FSTs). Among all the previous studies this one is more sophisticated in terms of integrating multiple information sources and formulating the problem as a search task within a unified framework. The evaluation results showed a diacritic error rate of 12.79% and a word error rate of 23.61% on Arabic Treebank corpus using FST with a Katz language modeling (LM).

Ananthakrishnan et al. [35] worked on diacritization with the goal of improving automatic speech recognition (ASR). They used a word-based language model using both diacritized and undiacritized words in the context but backed off to a character-based model for unseen words. They consulted Buckwalter [56] Arabic Morphological Analyzer (BAMA) to narrow possible diacritizations for unseen words but BAMA does not provide much improvement used in this manner.

Elshafei et al. [124] used HMM approach to solve the problem of automatic generation of the diacritic marks of the Arabic text. They focused on the issue of adding diacritics ‘Tashkeel’ to undiacritized Arabic text using statistical methods for language modeling. The approach requires a large corpus of fully diacritized text for extracting the language monograms, bigrams, and trigrams for words and letters. The word sequence of undiacritized Arabic text is an observation sequence from a Hidden Markov Model, where the hidden states are the possible diacritized expressions of the words. The optimal sequence of diacritized words (or states) is then efficiently obtained using Viterbi Algorithm. An evaluation of the basic algorithm is performed using the Quran text, and they reported an error rate of about 4.1% without using any knowledge-based tools like morphological analyzer.

Zitouni et al. [125] presented a complex statistical model for Arabic diacritic restoration. The approach is based on the Maximum entropy framework, which gives the system the ability to integrate different sources of knowledge. This model has the
advantage of successfully combining diverse sources of information ranging from lexical, segment-based (tokenizer) and part-of-speech tag (POS) features. In order to simulate real world applications both POS and segment-based features are generated by separate statistical systems. The segment-based features are extracted from a statistical morphological analysis using WFST approach and the POS features are generated by a parsing model using maximum entropy framework. They then use segment n-grams, segment position of the character being diacritized and the POS of the current segment along with lexical features including letter and word n-grams. Using a publicly available corpus (LDC’s Arabic Treebank Part 3), they achieved a diacritic error rate of 5.1%, a segment error rate 8.5%, and a word error rate of 17.3%. In case-ending-less setting, they obtained a diacritic error rate of 2.2%, a segment error rate 4.0%, and a word error rate of 7.2%.

Habash and Rambow [126] presented a diacritization system for written Arabic which is based on a lexical resource. It combines a tagger and a lexeme (prefix, stem, and suffix) language model (LLM). In this approach, a set of taggers are trained for individual linguistic features which are components of the full morphological tag such as core part-of-speech, tense, number including case, mood, nunation (Arabic: تَنْوُئ tanwīn) as additional features because of its importance to diacritization. In Arabic, they have used 2,000 to 20,000 morphological tags and the Buckwalter Arabic Morphological Analyzer (BAMA) is consulted to produce a list of possible analyses for a word. BAMA returns all possible morphological analyses including full diacritization for each analysis. The results of the individual taggers are used to choose among these possible analyses. The applied algorithm they proposed in Habash and Rambow [127] for choosing the best BAMA analysis that simply counts the number of predicted values for the set of linguistic features in each candidate analysis. For training their classifiers they used the exact training set defined by Zitouni et al. [125], a subpart of the third segment of the Penn Arabic Treebank (ATB3-Train) of 288,000 words. They have achieved best results with Tagger-LLM-3 configuration having word error rate (WER) of 14.9% and diacritic error rate (DER) of 4.8%.

Kübler and Mohamed [128] confronted vocalization as a classification problem in which they have to decide for each character in the unvocalized word whether it is followed by a short vowel or not. They investigated the importance of different types
of context. For classification they used a memory-based learner, *Tilburg memory based learner* (TiMBL) proposed by *Daelemans et al.* [129]. The results show that the word internal context provides enough information for vocalizing a high percentage correctly. The best parameter and feature setting results in an error rate of 6.64%, which is more than the results presented by *Zitouni et al.* [125] even though this system did not have access to either word segments or POS tags. Further more, adding lexical context as additional features did not increase the performance of the memory-based classifier TiMBL. Interestingly, the most informative feature is the character following the focus character, although in general the left character context within the focus word is more informative than the right character context. The learning curve shows that at least in the experiments with features only from within the focus word, a training set of 700,000 characters is sufficient for reliable results.

*Shaalan et al.* [130] developed a hybrid approach for building Arabic diacritizer. The system combination has Lexical Retrieval (LR), Diacritized Bigram (DB), Support Vector Machines (SVM), and case ending (CE) module. The LR and DB techniques are lookup processes. The SVM approach is used to tokenize and automatically annotate tokens with the correct POS tags. Then by searching the Arabic lexicon using a token and the corresponding POS, the correct diacritization result can be reached, even though multiple ambiguous words are retrieved from the lexicon. They have used the Buckwalter's morphological analyzer after removing all case ending diacritics from the suffixes table in order to prevent the generation of the case ending output. This is because the case ending diacritics are deliberately handled syntactically using SVM with Penn Arabic Treebank (ATB). This hybrid approach works as a combination with assigned priority from highest to lowest order as LR, DB and SVM respectively. The best evaluation results of hybrid system LR+DB+SVM+CE has shown the word error rate (WER) of 17.31% and diacritic error rate (DER) of 4.41%.

*Rashwan et al.* [131] presented a two-layer stochastic system to automatically diacritize raw Arabic text. The first layer tries to decide about the most likely diacritics by choosing the sequence of full-form Arabic word diacritizations with maximum marginal probability via long A* lattice search and n-gram probability estimation. Fully non factorizing statistical methods working on full-form words are faster to learn but suffer from poor coverage (OOV) which can be complemented by linguistic factorization analyzers. The second layer factorizes each Arabic word into
its possible morphological constituents such as prefix, root, pattern and suffix, then
uses n-gram probability estimation and A* lattice search to select among the possible
factorizations to get the most likely diacriticizations sequence. After evaluation of the
hybrid diacriticizer at a large (3,250K words) training data the best results for
morphological and syntactic (case ending) diacritization error rates are 3.6% and
13.0% respectively. It has shown a marginal gain with morphological diacritization.

Haertel et al. [132] developed an automatic diacritization for low-resource
languages using a hybrid conditional Markov model (CMM). The diacritization of
three Semitic languages, namely Syriac, Arabic and Hebrew is performed using
only respective diacritized texts. The number of diacritics considered in Syriac,
Arabic and Hebrew are 9, 8, and 17 respectively. Unlike most of the previous
methods that required the use of lexical tools such as POS taggers, segmenters,
morphological analyzers and linguistic rules to produce good results, they have
modeled a low-resource, data-driven, and language-independent approach using a
hybrid word and consonant-level CMM. After evaluating the system on Penn Arabic
Treebank (test size 51,664 tokens) the best results for Arabic language are reported
as word error rate (WER) of 15.02% and diacritization error rate (DER) of 5.15%
with case endings and without using any morphological analyzer. Similarly for
experiments in Syriac, they used the New Testament portion of the Peshitta (test size
11,021 tokens) and best evaluation results has shown a WER of 10.54% and DER of
4.29%. The Hebrew Bible is used for training a testing (with test size of 49,455
tokens) and best evaluation results in this case are 22.18% (WER) and 10.71%
(DER).

2.2.4 Related Work

Most prior work in Arabic-related transliteration was developed for the purpose of
machine translation. Due to the morphological complexity of the Arabic language,
much research has focused on the effect of morphology on Arabic Information
Retrieval (IR) [53]. In particular, we have found just a single paper addressing the
same problem of transliteration as ours. In the paper titled Punjabi Machine
Transliteration (PMT) Malik, [133] demonstrated a very simple rules based
transliteration system for Shahmukhi-to-Gurmukhi script. Firstly, two scripts are
discussed and compared. Based on this comparison and analysis, character mappings
between Shahmukhi and Gurmukhi are drawn and transliteration rules are formed.
Along with this some dependency rules have been formed for special characters like aspirated consonants, non-aspirated consonants, Alef (א), Alef Madda (אָ), Vav (ו), Choti Yeh (י) etc. The primary limitation of this system is that this system works only on input data which has been manually edited for missing vowels or diacritical marks (*the basic ambiguity of Arabic script*). As the author himself admitted that the accuracy of the PMT system depends upon the supplying necessary diacritical marks (manual work) and absence of the necessary diacritical marks affects the accuracy of PMT greatly.

**Observed Limitations of PMT System:**

1. It is a simple system limited to rules based model only.
2. The basic but critical problem of missing diacritics is NOT considered.
3. Absence of the necessary diacritical marks greatly affects the accuracy of PMT.
4. They have not utilized any corpus help.
5. Statistical approaches like word frequency, Bi-gram, Tri-gram or N-gram etc. are not used.