Soft Computing (2020) 24:16057–16079
https://doi.org/10.1007/s00500-020-04923-7

METHODOLOGIES AND APPLICATION

A forefront to machine translation technology: deployment on the cloud as a service to enhance QoS parameters

Muskaan Singh[1] · **Ravinder Kumar**[2] · **Inderveer Chana**[2]

Published online: 28 April 2020

Abstract
Machine translation system (MTS) constitutes of functionally heterogeneous modules for processing source language to a given target language. Deploying such an application on a stand-alone system requires much time, knowledge and complications. It even becomes more challenging for a common user to utilize such a complex application. This paper presents a MTS that has been developed using a combination of linguistic rich, rule-based and prominent neural-based approach. The proposed MTS is deployed on the cloud to offer translation as a cloud service and improve the quality of service (QoS) from a stand-alone system. It is developed on TensorFlow and deployed under the cluster of virtual machines in the Amazon web server (EC2). The significance of this paper is to demonstrate management of recurrent changes in term of corpus, domain, algorithm and rules. Further, the paper also compares the MTS as deployed on stand-alone machine and on cloud for different QoS parameters like response time, server load, CPU utilization and throughput. The experimental results assert that in the translation task, with the availability of elastic computing resources in the cloud environment, the job completion time irrespective of its size can be assured to be within a fixed time limit with high accuracy.

Keywords Machine translation system · Amazon web server · Elastic computing unit · Quality of service

1 Introduction

Soft computing (SC), introduced by Yager et al. (1994), is endeavoured to provide the imprecision of the real world (Zadeh 1996). It provides a reliable solution for complex problems such as estimation with adequate precision (Naderpour and Mirrashid 2020b). It comprises of three models fuzzy systems, artificial neural network (ANN) and optimization algorithm. The ANN model computations are inspired by the biological neural network of the brain. It provides approximate solutions for a nonlinear function as it comprises of several neurons as nonlinear components (Siddique and Adeli 2013). It has applications in various diverse areas such as control (Kim et al. 2019), classification (Naderpour and Mirrashid 2019), biology (Dunn et al. 2019), construction (Naderpour et al. 2018), secure data aggregation and efficient data processing in the large-scale wireless sensor network (Shobana et al. 2020), remote sensing image classification of natural terrain features (Kundra and Sadawarti 2015), electric load forecasting (Zhang and Hong 2019; Dong et al. 2018) and forecasting the motion of a floating platform (Hong et al. 2019). Fuzzy systems are based on the ability of human brain reasoning. To determine the fuzzy output of nonlinear system provided with an input vector uses a rule-base (Gorzalczany and Gluszek 2002). These techniques can be merged to form a hybrid model such as a neuro-fuzzy model. This hybrid model is used in several application areas such as strength (Naderpour et al. 2019), earthquake (Polykretis et al. 2019), power amplifier (Galaviz-Aguilar et al. 2019), security threats detection (Neelaveni and Sridevi 2019), speech recognition (Asemi et al. 2019), machine translation (Singh et al. 2019b) joints (Naderpour and Mirrashid 2019) and columns (Naderpour and Mirrashid 2020a). The computational intelligence problems also involve soft computing approach to solve such complex problems (Naderpour and Mirrashid 2020b). One such computational intelligence problem is natural language processing (NLP); it involves

Communicated by V. Loia.

✉ Muskaan Singh
muskaan_singh@thapar.edu

[1] Language Technology and Machine Learning Research Lab, CSED, Thapar Institute of Engineering and Technology, Patiala, Punjab, India

[2] CSED, Thapar Institute of Engineering and Technology, Patiala, Punjab, India

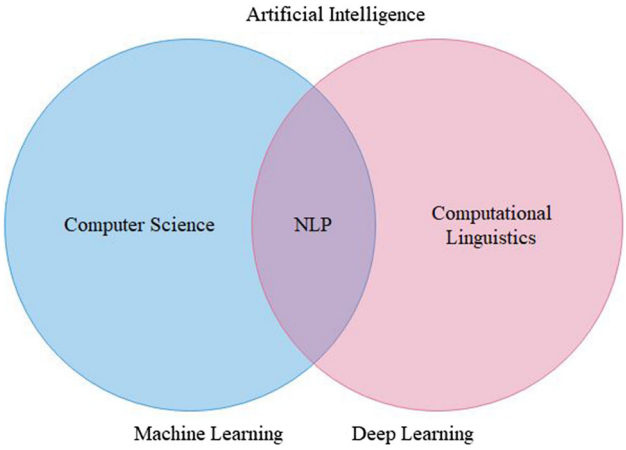

Fig. 1 Natural language processing

computers to understand and process the text in natural language such as German, French, Sanskrit, Hindi, Tamil and Telugu. It is a multidisciplinary field involving computer scientist and computational linguistics as shown in Fig. 1. The goal of NLP is to build a computational model for its analysis and generation. NLP involves technological motivation for developing intelligent computer system such as machine translation, sentiment analysis, information extraction, question answering system and speech recognization as exhibited in Fig. 2. There is also cognitive and linguistic motivation for processing natural language. It involves natural language understanding and its generation as in Fig. 3. Machine translation (MT) is one of the many applications of NLP (Bharati et al. 1995; Chowdhury 2003). MT is a process of translating source language to target language using computing technology. Human translators or editors can be involved in the process of MT, although minimal human aid is the goal of MT. The field of man–machine interaction and artificial intelligence involves the processing of natural language. A translation provided by the MTS involves both syntactic and semantic aspects of language to ensue for the correctness of the translation. Cloud computing is a model which enables ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management efforts or service provider interaction (Fox et al. 2009). Clouds are classified based on service type or deployment type (Foster et al. 2008). It consists of three service models SaaS, PaaS and IaaS (Ahmad and Ranka 2016; Rimal et al. 2009).

1.1 Need of MT deployment on cloud

Numerous applications are being deployed on cloud nowadays as listed in Table 1. It includes MT application offered through the cloud and its outcome achieved by reducing the deployment time and auto-scaling of applications.

Many factors are governing the need for MTS deployment on the cloud such as deployment faster and easier, handling of frequent updates, manageability, speed, reliability, quality of service (QoS), enhancement, rapid provisioning, elasticity and resilience also exhibited in Fig. 4. An experiment on deploying MT system (Hindi to Punjabi) on the cloud was conducted. It took a book of 70 pages to translate on stand-alone machine 71 min, whereas on the cloud it turned down to 5 min (Kumar et al. 2013c). Hence, deployment of MT systems on the cloud is a viable option for future users.

1.2 Problem statement

MT is a highly compute-intensive application including less amount of input data but more processing. Therefore, such applications when processed on a simple computer do not offer adequate performance and accuracy of the application. MTS consists of various heterogeneous modules which comprise of large code blocks. It is a complex NLP application containing heterogeneous modules. Deployment of such a complex system on the stand-alone system is cumbersome, time-consuming and knowledge intensive. It takes hours to load, configure and run the system. To overcome, MTS can be configured on cloud infrastructure to exhibit better accuracy and performance. This will facilitate the auto-scaling of computational resources for changing load conditions (Ahmad 2013). The deployment on cloud to deliver Quality of Service (QoS) to the end-user.

1.3 Contribution

Neural machine translation (NMT) has been outperforming traditional statistical phrase-based and RBMT (Singh et al. 2019a). Driven by its performance, we implemented a neural model by training linguistic rich features drawn from the rule-based system for Indian language pair, i.e. Sanskrit–Hindi. After the development of MTS, we build a website for a better user experience. Later, the web interface was deployed on the cloud to provide the MTS as a service.

- Initially parallel corpus was gathered which was available from different sources and rest of the data was manually created for building a neural model using encoder–decoder architecture with attention mechanism (Ahmad et al. 2011a).
- We trained and tested the NMT system with different models, activation functions, training data, epochs, sentence lengths to yield better accuracy of the proposed system.
- We then merged linguistic tools output from the classical rule-based approach as features embedding matrices

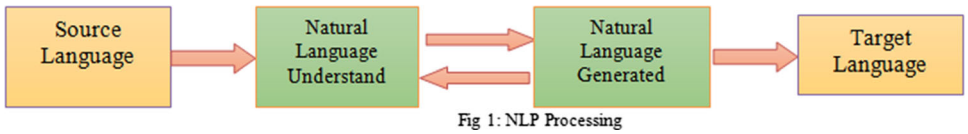

Fig. 2 Natural language processing involving translation

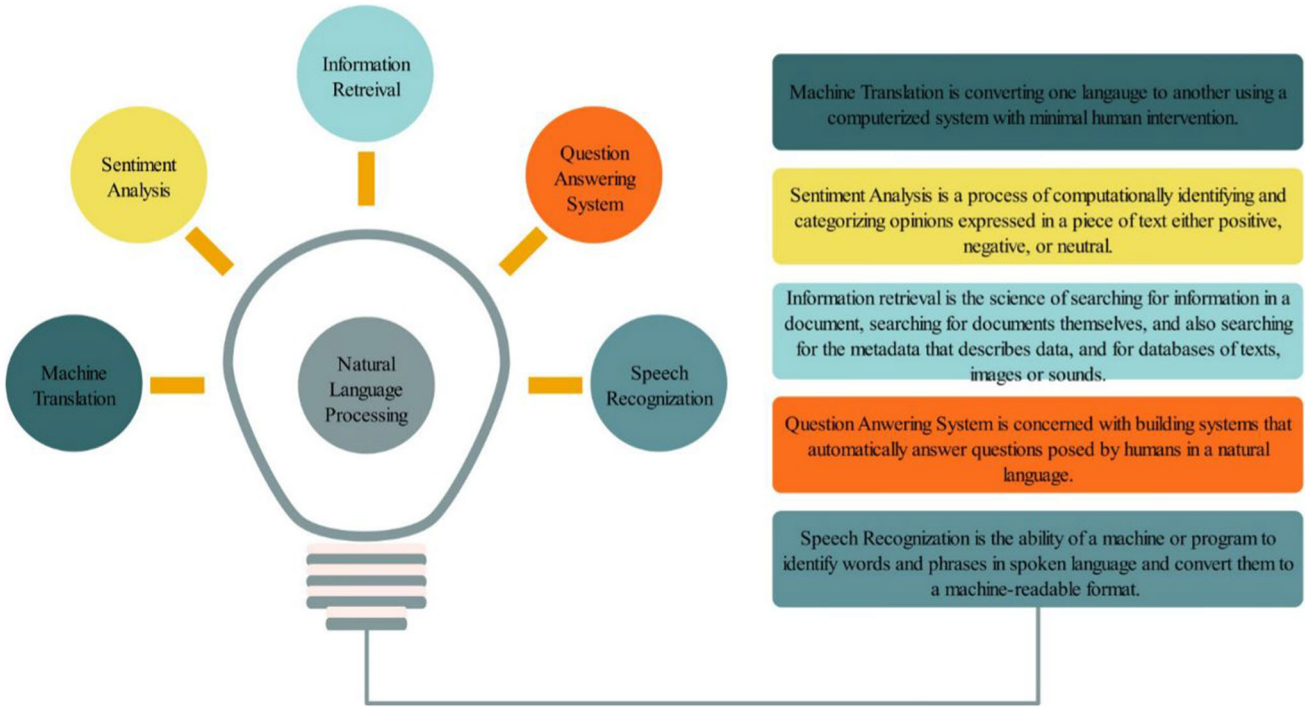

Fig. 3 Natural language processing applications

in NMT to test whether it guides for the disambiguation translation of words as the same word may have a different meaning in a different context. We also tested whether it reduces the data sparseness and performs meaningful tokenization.
- An web interface was developed for better user-interface.
- The web interface was deployed on AWS cloud with EC2 which eases time, knowledge and complications. It even becomes challenging for a common user to utilize such a complex application. It elevates the MTS performance by managing recurrent changes in term of either corpus, domain, algorithm and rules.

As the workload increases changing the deploying environment is needed for such complex compute-intensive applications. It improves throughput, response time and reduces deployment time. These systems are offered by both academic and commercial organizations and have been providing services to the end-users.

1.4 Paper organization

The remaining of the paper is designed with review of the existing state-of-art literature in Sect. 2. Section 3 demonstrates the proposed Sanskrit to Hindi MTS as a Service. Section 4 describes its experimental details along with its results in Sect. 5. Lastly, the paper is concluded in Sect. 6.

2 Review of the existing work

2.1 MTS on cloud

The MTS development is prominently using deep neural network-based approach It has been producing significant improvement in the results after statistical MT. These systems deployed on the cloud provide a scalable service with various benefits. Some of the translation system earlier deployed on the cloud has been discussed in this section. Kiyurkchiev et al. (2019) developed a DisPeL learning portal which is an e-learning portal for learners and tutors. It is implemented using a combination of Angular 4 and a proprietary SPA framework based on jQuery and vanilla JavaScript. It provides better scalability and cognitive services like automatic translations and search. The advanced reporting instruments via dedicated reporting databases and PowerBI. The application skeleton is developed in NET Framework 4.6. The system

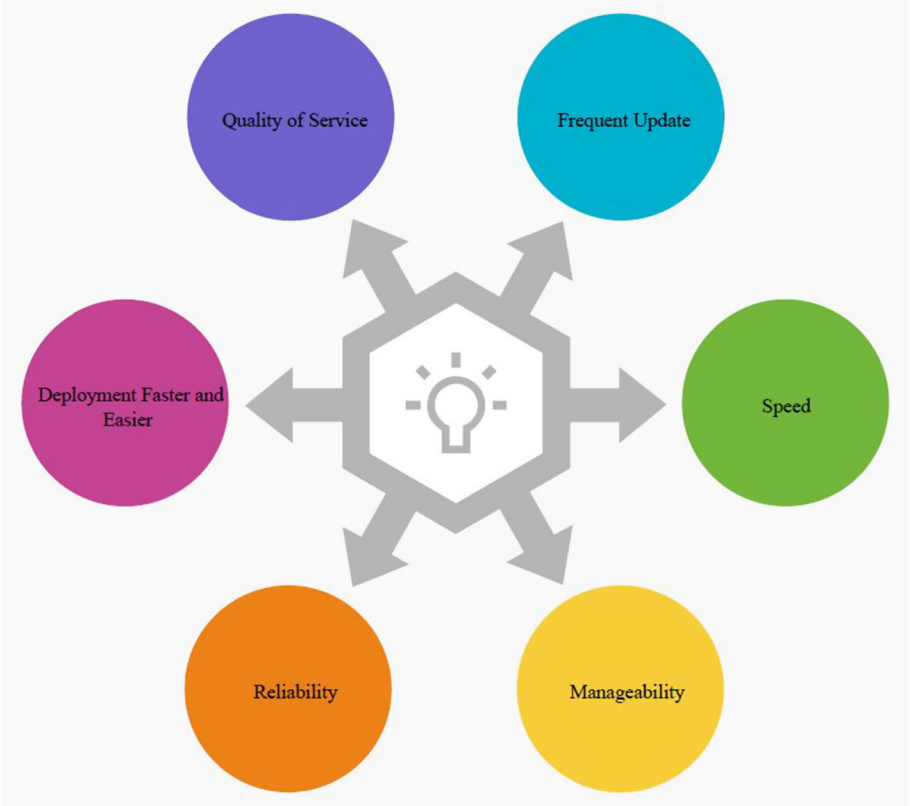

Fig. 4 Need for cloud computing in machine translation systems

Table 1 Existing deployment of applications on clouds

Domain	Specific application	Outcome achieved
Natural language processing	Machine translation system (Kumar et al. 2013c)	→ Deployment time reduced
		→ Provision for auto-scaling of application
Cloud-based application	3-D application (Shi et al. 2011)	→ Virtual graphics processing optimized
		→ Cloud-based framework for 3-D virtual appliance
Geospatial research	Model processing (Schwartze et al. 2011)	→ It takes task of model processing field
		→ Open source software is used
Operating system	File system related development (Abd-El-Malek et al. 2012)	→ Virtual machine can be migrated
		→ Cache for unified buffering
		→ File system is portable
Educational field	Teaching operating system (Laadan et al. 2010)	→ Helpful to students for assignment, as can be deployed on the students computer
		→ Can be practised without the facility in laboratories and gives kernel level project experience
Server dependencies	Legacy distributed system (Machida et al. 2010)	→ Generates graphs with dependency representation
		→ Optimum deployment on server framework

provides a quick implementation of report generator job as another type of application, supported by NET Core. Huang et al. (2019) provides a scalable model-parallelism library for training giant networks. The single model achieved 84.4% and top-5 validation accuracy with single-crop 97%, the re-computation time 23%, load imbalance 3.2% and minimizes the performance overhead. Chen et al. (2019) provides a fully synthesizable C++ template library with the considerations of

FPGA implementation. An optimized system configuration is developed to maximize overall performance. The software stack is merged with the DNN accelerator, to provide a complete system-level solution for the users who need acceleration services. Vaswani et al. (2018) developed a library by Google Brain and DeepMind. This library of deep learning models and datasets designed to make deep learning research faster and more accessible Tensor to Tensor uses TensorFlow throughout the implementation. It benefits the researcher to train models on CPU, GPU (single or multiple), and TPU, locally and in the cloud, usually with no or minimal device-specific code or configuration. There have been a strong focus on performance as well as usability. The future work may reduce this scaling factor of encoder and decoder model. Venugopal and Zollmann (2009) proposed an end-to-end grammar-based statistical MTS running on Hadoop. This system was able to scale to build grammars for large scale in reasonable time frames. Ferrández-Tordera et al. (2016) proposed CloudLM, an open-source cloud-based language model intended for MT with distributed architecture. This proposed system is stable, robust and is very efficient. Ahmad et al. (2011b) proposed a Map reduced based framework for machine translation which enhances the throughput of the system. Kumar et al. (2013b) proposed a MTS using MapReduce Framework to assure QoS. With the experimental analysis, they show that job completion time for any translation is within a fixed time limit irrespective of its size. There are a few others MTS proposed with cloud-based framework (Ahmad et al. 2011a; Kumar et al. 2013a, c; Vasiļjevs et al. 2011) to achieve better performance for current MTSs. Another experiment of deployment by Ahmad et al. (2014) using Storm, a distributed computing framework is used for deploying on cloud. It reduces job completion time, gives very good user experience by web interface and response time. The elapsed time for a translation job after the submission of the job is 4 s. In a future modification of the storm topology to apply parallelism at word level processing, different MT systems developed on the cloud have been compared based on their techniques used, technical specifications employed and the outcome achieved with their limitations and future work. A comparative analysis of various cloud-based approaches for MT is presented in Tables 2 and 3.

2.2 MTS for the Sanskrit language

In this paper, Sanskrit–Hindi hybrid MT system is proposed where hybrid mechanism (a combination of linguistic and neural-based) will be applied for translation and cloud resources are used for processing and computation for translating Sanskrit to the Hindi Language. Sanskrit is one of the oldest Indo-European languages. In Uttarakhand, Sanskrit is used as their official language and it is also known as the mother of the other languages. This language was used by Hinduism, Buddhism and Jainism in their holy books, Vedas and other philosophical texts. One language helps to unite the world culturally, technologically and socially and MTS is used for the same purpose. A very few researchers developed systems for translation of ancient morphological rich language, i.e. Sanskrit. Early efforts included the use of a rule-based approach for translation of English–Sanskrit. Rathod and Sondur (2012) involves parsing of the sentence performed along with the module such as a token generator, vichcchceda, translator and in the final phase the translated output is converted speech output. For smaller and simple sentences, translation provided is correct. Complex sentences cannot be handled by the system. There is no information regarding the semantics of the text. This drawback was overcome by Barkade and Devale (2010) with the semantic mapper, translator, composer and lexical parser. Information can be obtained through introspection and analysis. Semantic analysis is considered in this system and provided by the system for translation. Accuracy of the entire system is dependent on each module. E-Trans system (Bahadur et al. 2012) based on formulation of context-free grammar. The results produced by this system were quite promising for small and large sentences of English to Sanskrit. For handling spoken translation, English–Sanskrit translation system and synthesizer (Rathod and Sondur 2012) is developed. It consist of various modules for processing such as a token generator, parser generator, RBMT/EBMT engine, bilingual database, text output and waveform generator. The research work is useful for sharing worldwide knowledge with Indian translation approach. Another system for speech translation, TranSish (Upadhyay Pankaj 2014) provides an interactive interface for people to interact with the Sanskrit language in speech form. It provides translation for the present tense only. Employing example-based approach ANN and rule-based model for English–Sanskrit translation (EST) were developed where they proposed detection rules using ANN mechanism for divergence related sentences and implemented some rules for its adaptation (Mishra and Mishra 2009). Feedforward ANN to make a selection of Sanskrit words and adjective from English to Sanskrit user data vector (UDV). Evaluation measures are used to calculate the performance of the system; BLEU provides 0.325 accuracies for a random sentence, precision, recall and F-measure provides 0.8, and METEOR provides 0.811. This system provides translation for an English sentence of only 6 types of sentence structure. The current research work integrates ANN with java. To improve typo errors, CBS (case-based reasoning) and data mining techniques can be considered to integrate for the future work. All these systems were developed for translating English–Sanskrit. There are very few attempts for translation of Sanskrit to Hindi. A proposed SaHit-statistical approach for Sanskrit–Hindi translation (Pandey and Jha 2016) where they trained their system on the platform—the

Table 2 Existing work of machine translation deployed on cloud

Research work	Year	Research organization	Stream	Methodology	Results	Limitation and future research directions
Kiyurkchiev et al. (2019)	2019	Plovdiv University	Statistical + Neural	DisPeL learning portal is a e-learning portal for learners and tutors Implemented using a combination of Angular 4 and a proprietary SPA framework based on jQuery and vanilla JavaScript	Better scalability DisPeL with cognitive services like automatic translations and searching Advanced reporting instruments via dedicated reporting databases and PowerBI	The application skeleton in NET framework 4.6 Quick implementation of report generator job as another type of application, supported by NET Core
Huang et al. (2019)	2019	NA	Neural network	A scalable model-parallelism library for training giant networks	Single model achieved 84.4% Top-5 validation accuracy with single-crop is 97%	Re-computation time 23%, load imbalance 3.2% and minimize the performance overhead
Chen et al. (2019)	2019	University of Illinois at Urbana-Champaign, USA	Neural Network	A fully synthesizable C++ template library with the considerations of FPGA implementation	An optimized system configuration to maximize the overall performance Software stack together with the DNN accelerator, to provide a complete system level solution for the users who need acceleration services	NA
Vaswani et al. (2018)	2018	Google Brain and DeepMind	Deep neural network	It has library of deep learning models and datasets designed to make deep learning research faster and more accessible T2T uses TensorFlow throughout	Researchers can train models on CPU, GPU (single or multiple) and TPU, locally and in the cloud, usually with no or minimal device-specific code or configuration A strong focus on performance as well as usability	Future work may reduce this scaling factor of encoder and decoder model
Ferrández-Tordera et al. (2016)	2016	ADAPT Centre, School of Computing, Dublin City University, Ireland	Statistical	Language Model (LM) in Apache Solr Cloud-based LM is in Moses Efficiency enhancement by cache and block query	Cache reduces 70% memory usage Memory used argument by 20% Sentences decoding 100 sentences reduce the time by using cache 89% and memory used increments by 195% Block queries when decoding 1 sentences (9% faster) and slower for 10 to 100 sentences	Enhancing the efficiency and time by keeping the connection alive between the Moses and Solr (so that a new query does not need to reopen the connection)
Ahmad et al. (2014)	2014	LTRC, IIIT Hyderabad	Statistical	Storm, a distributed computing framework is used for deploying on cloud	It reduces the job completion time It gives very good user experience by web interface Response time The elapsed time for translation job after the submission of the job is 4 s	In the future modification of the storm topology to apply parallelism at word level processing

Table 2 continued

Research work	Year	Research organization	Stream	Methodology	Results	Limitation and future research directions
Kumar et al. (2013c)	2013	IIIT Hyderabad, India	Transfer	CentOS 5.3 as guest O.S Xen is used for virtualization of hardware 5.7 as Host operating system Hadoop as middleware for word load partitioning Eucalyptus for setting up the cloud infrastructure	To translate a book in stand-alone system took 71 min and 25 s and on eucalyptus, cloud environment took 5 min and 27 s It requires 3 min to scale up the MT application after provisioning the new resources in the system	Enhancing the virtual appliance so that it is available as repositories and from there can be deployed in the cloud that would enable MT system to handle frequent updates as well
Kumar et al. (2013a)	2013	IIIT Hyderabad, India		Sampark machine translation system is used Hadoop, open source implementation of Mapreduce for partitioning jobs Eucalyptus cloud for running the cloud system For virtualization CentOS (5.7) as host O.S with Xen	A cluster of 5 nodes for 25,600 sentences total time to compute is 12,920 and throughput is 119 per minute If the partition size is doubled the throughput increases by less than 5%	Extension of the approach for QoS to various NLP applications that exhibits list homomorphism and can be partitioned on distributed computing
Skadiņš et al. (2012)	2012	University of Edinburgh, Zagreb, Copenhagen, Uppsala, Moravia	Statistical	MT training and decoding tool Moses is used	Achieved better quality, reputation, convenience and domain-specific translation for user-specific text Ready resource for studying and teaching purpose for academic institutions	Support for only some specific format is provided in this version of the system For scalability, the system is depended on cluster hosted on Amazon Web service infrastructure
Vasiļjevs et al. (2011)	2011	Moravia, Semlab and University of Edinburgh	Transfer	SMT training and decoding toolkit on Moses Morvia used LetsMT platform to train and evaluate SMT systems for polish and Czech	The system has to lead a strong increase in translator productivity	Depended on translation memories for parallel data Translators receive translations suggestions provided by the selected MT engine running on LetMT!
Gao and Vogel (2010)	2010	Carnegie Mellon University, USA	Phrase based	Chaski toolkit is run on the top of the Hadoop framework GIZA++ and Moses are used for training data and aligning data	The training time of word alignment is reduced from 47 to 8 h Time for phrase extraction from 21 to 43 min The output phrase table is compatible with the Moses decoder	Distributed word alignment technique used has no changes on performance Chaski toolkit used and Moses phrase table has no significant difference Language model focus on the validity of results rather BLEU score
Venugopal and Zollmann (2009)	2009	Carnegie Mellon University, USA	Statistical	Syntax augmented machine translation (SAMT) on mapreduce Hadoop for parse tree of the target language	Translation quality measured with IBM-BLEU % for 67M it took 25.88 min and for 230M it took 26.28 min	Using SAMT (syntax-based augmented grammar), the system exhibits even though consistent but small improvements

Table 3 Existing commercial MTS: a comparative study

Commercial system	Developer	Platform	Language pair	Approach	Findings
Bing translator	Microsoft	Web application	52	Statistical	It is a web service with back-end translation software For translating an entire web page 4 bilingual viewer layouts are present
Microsoft translator	Microsoft	Windows, Apple and Android phone	60 (for text) and 9(for speech)	Statistical	Available for both personal use and business use
IBM Translator	IBM	Web service	5 (conversational domain), 10 (news domain), 5 (patent domain)	Hybrid	It is a translation service with multi-domain translations in real time
SDL language weaver	SDL Trados, Germany	Web application	111+	Statistical	It is computer-assisted translation translation software suit
Slate	Precision Translation tools	Windows, Linux (i86-64)	29	Statistical	It is personalized translation engine outperforming cloud-based MT without their drawbacks requiring less amount of time
Kantan MT	Xcelerator Machine Translation Ltd	Web application, Windows, MAC, Linux	59	Statistical	An high-quality MT service is provided using the power and flexibility of cloud
Xerox Easy Translator	Xerox Easy Translator Service	Web application	57	Hybrid	It offers professional translation service through multiple access points, offering flexible, secure and affordable services for translation

Microsoft Translator Hub (MTHub) for only simple Sanskrit prose texts. This system takes input only in Devanagari Unicode Script and gives output in same. Sometimes the system does not response to long and compound sentences. In this work, MTS is proposed which handles all the limitations of the existed MTS developed for the Sanskrit language.

3 Sanskrit–Hindi hybrid machine translation system (SHH-MTS) as a service

The MTS improves performance through recurrent changes in term of either corpus, domain, algorithm and rules. These rapid changes are very difficult to be updated in stand-alone MTS. To facilitate this problem, we have proposed that it must be deployed on cloud to provide quality of service (QoS) to the end-user. Firstly, Sanskrit–Hindi hybrid machine translation system (SHH-MTS) architecture is designed and developed. Later, post-development it is deployed on cloud to test and analyse its performance on various parameters such as throughput, server load, response time and CPU utilization.

3.1 Characteristics of Sanskrit–Hindi machine translation system

There are different characteristics listed for Sanskrit–Hindi machine translation system (SHH-MTS)

- The SHH-MTS developed is the integration of linguistic rich features with prominent result-oriented approach, i.e. rule-based and neural-based which is gaining significant attention nowadays.
- As source language, Sanskrit is a linguistically rich language having old scriptures like Vedas written in this language, which are not accessible and understandable to general people. Through SHH-MTS, the old scriptures can be easily accessed in other languages.
- The proposed system also benefits the teaching–learning process by assisting with Sanskrit content. The system aids the students by illustrations of grammatical information for the Sanskrit text such as parts-of-speech tagging, parsing, Sandhi-splitting, word sense disambiguation and relations between different words of a sentence.
- The system is readily updated with recurrent frequent updates of performance in the term is corpus, domain, algorithm and rules.
- The SHH-MTS is deployed on the cloud and provided as a service. It enhances the deployment making it easier to perform and easy to use by the common user as no pre-requisite or knowledge of NLP required.
- Auto-tuning for neural-based MT used in our proposed system is not possible at the local host due to memory issues, but it is possible on the cloud. Several layers are added automatically to attain maximum accuracy and high speed.

The following sections contain the details of entire SHH-MTS developed in phases, i.e. different linguistic tools output feed for embedding as features in the encoder of neural-based encoder–decoder architecture which trains the systems for learning and predicting the translation of Sanskrit words into Hindi words using linguistic features.

3.2 SHH-MTS: rule-based machine translation system; extraction of linguistic features

The rule-based system follows pipeline architecture, each module or tool has been described in this section. Different linguistic tools have been used as input features in neural machine translation (NMT) to train system more efficiently. Some linguistic features are language specific, i.e. Sandhi-splitter. These tools are mostly rule-based developed under Project funded by MIT. We have also developed a web interface for better user experience using Anusaaraka Engine (Chaudhury et al. 2010) for providing translation to common users.

- Pre-processing of user input: It takes input from user, cleanses, normalize the text, converts the input notations into WX notation, call and invokes MT system which performs computation and shows the output result.
- Tokenizer: Tokenizer receives a flow of character and that character breaks into individual words called as tokens (words, punctuation, markers). It removes the formatting information and add a sentence tag. Here, the term morphology is used for linguistics. It refers to study of words, their internal process and their word meaning. The model has stream of words those words are tokenized first and then morphology gives meaning to those words (Pappu and Sanyal 2008).
- Sandhi-Splitter: It is invoked only when input text contains Sanskrit sandhied words. It splits these words as well as compound words (Sachin 2007; Kumar et al. 2010).
- Morphological Analyser: It splits into roots and grammatical suffixes. There are different units, and one of each unit provides meaning as well as grammatical function. It also provides inflectional analysis, prunes the answer, uses local morph analysis to handle unrecognized words and produces derivational analysis of the derived roots (Bharati et al. 2006; Mittal 2010; Jha et al. 2009).
- Parsing: Parser is used as compiler and interpreter that break data into smaller units for easy translation of one language to another. Parsers takes input from the

sequence of words or tokens those inputs are translated in the form of parse tree or an abstract syntax tree. It converts the source language to target language in the form of tree like noun, verbs and attribute. It produces morph analysis according to context along with karaka analysis. According to computational Paninian grammar, it identifies and names the relation between verb and its participants (Kulkarni et al. 2010; Goyal et al. 2009; Kulkarni and Kumar 2011; Kulkarni and Ramakrishnamacharyulu 2013; Kulkarni 2013).

- Shallow Parsing: If the parser fails on any input, it does minimum parsing of the sentence and produces pruned morph analysis to the next layer (Huet 2006; Kumar et al. 2010; Kulkarni et al. 2010).
- Word Sense Disambiguation (WSD): The modules performs word sense disambiguation of input sentence words roots, vibhakti and lakara. Identifies correct sense of a Sanskrit word (Bharati and Kulkarni 2007).
- Parts of Speech Tag (POS): It adds parts of speech tags to each word such as adjective, verb or noun. tags (Hellwig 2009, 2010).
- Chunker: This phase performs minimum grouping of words in a sentence such as noun phrase, verb phrase, adjective phrase. The rule base allocates a appropriate chunk tag to it. Nandi and Ramasree (2013).
- Hindi Lexical Transfer: The Sanskrit Lexicon is transferred to Hindi identifying root words using dictionary. The output is formatted according to Hindi Generator, which generates the output in Hindi Language corresponding to the Sanskrit language. This module also performs transliteration in case translation fails (Kumar et al. 2009).
- Hindi Generator: This phase involves sentence-level generator performs agreement between noun, adjective and verb in the target language; addition of vibhakti markers 'ne' and dropping 'ko' at required positions. Final generation involves root words and their associated grammatical features, corresponding suffixes and concatenates them and also concatenates the generated words into a sentence (Bharati and Kulkarni 2007).

Hence, translation of each Sanskrit word to its corresponding Hindi word is performed using linguistic rules and tools. Further, these data are passed to consequent phase. The data passed to the next phase are converted into comma-separated values (CSV) format suitable for training, model development and fitting the values for neural-based encoder–decoder architecture for predicting translation of Sanskrit word to Hindi word. These linguistic tools output is embedded as features for input encoding of source sentence as they help better in disambiguation of words.

3.3 SHH-MTS: neural network-based RNN approach

The encoder–decoder recurrent neural network consisting of encoder reading a variable length input sequence and decoder predicting a variable length output sequence. The dense output layer is used to predict each character in the output sequence in one time rather recursively during training. Firstly, we define a model and once it fits it can be used to make translation predictions. The model defined for training has learned weights for this operation, but the structure of the model is not designed for calling recursively to be generate one character at a time. The encoder model takes input layer from the encoder in the trained model and gives output as hidden layer and cell state tensors. On the other hand, decoder needs hidden layer and cell state from encoder as initial state for the model defined. Both the encoder and decoder will be called recursively for each character that is to generated in the translation sequence (Cho et al. 2014).

The neural network encoder–decoder architecture with bidirectional RNN (Bahdanau et al. 2014) is implemented for predicting Hindi translation corresponding to Sanskrit translation. It consists of grated recurrent units (GRU) for computation. The implementation (Bahdanau et al. 2014) consists of input sequence feed with linguistic features (Sennrich and Haddow 2016). Given a source sentence $x = (x_1, x_2, \ldots, x_m)$ is read and computes hidden states for forward direction $(\vec{h}_1, \vec{h}_2, \vec{h}_3 \ldots \vec{h}_n)$ and for backward states $(\overleftarrow{h}_1, \overleftarrow{h}_2, \ldots, \overleftarrow{h}_n)$. The detailed computations have been displayed in Algorithm. Both of these forward computation and backward computation are then merged to form an annotation vector (h_i) as explained in Algorithm 2. The encoder input was a combination of linguistic features formed as feature embedding matrices as computed in Algorithm 3. The decoder further predicts the target sequence $y = (y_1, y_2, \ldots, y_n)$ based on context vector (c_i) computed in Algorithm 4 from weighted sum of annotations h_i, recurrent hidden state (s_i) and previously computed word y_{i-1}. The alignment model (a_{ij}) models the probability that x_j is aligned to y_i or not as in Algorithm 5. It is feedforward single layer network learned through back-propagation. The output is predicted using learned distribution. The implementation was carried out with Tensor flow (Abadi et al. 2016) at back end with Keras (Ketkar 2017) using encoder–decoder architecture for developing the system. In addition, we have considered the state constraints in order to minimize the computational complexity and generalized the results inspired by the work (Sun et al. 2018).

$$d_{i,iz} = f_i(w_s) + F_{i,z}(W_s, \bar{W}_s - 1) \tag{1}$$

where $i = 1, \ldots, n$, $ji = 1, \ldots, s_i - 1$, $s_i > 1$,$n > 1$ and both s_i, n are positive integers.

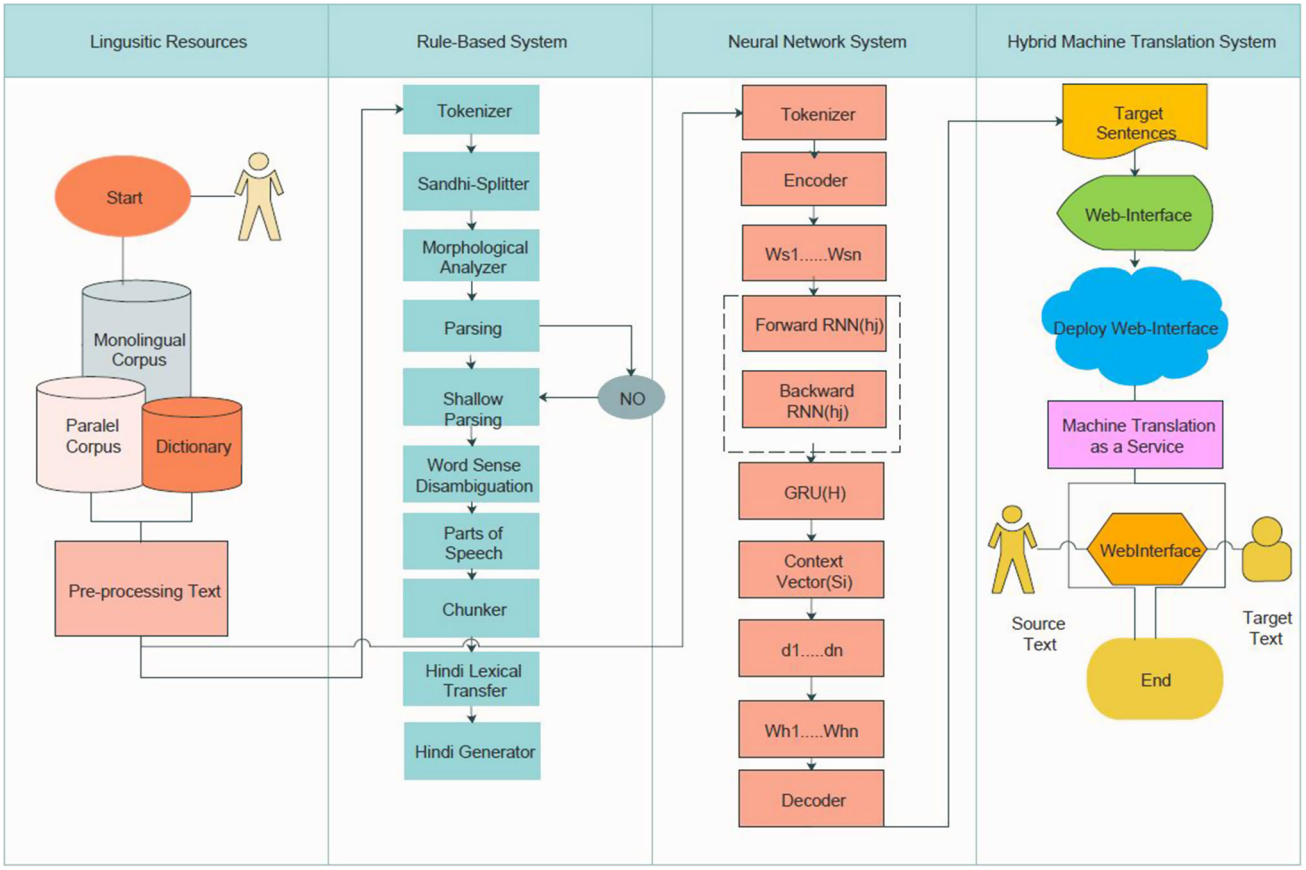

Fig. 5 Flow chart of proposed system

$$d_{i,ji} = d_{i,ji+1} + F_{i,ji}(\bar{d}), ji. \tag{2}$$

Here, $d = [d_i, ji, \ldots, \bar{d}_{i,ji}]^{\mathrm{T}}$ represents the states of encoder and decoder mechanism. The $\bar{d}_{i,ji} = [d_{i1}, \ldots, d_{i,ji}]^{\mathrm{T}} \in R_{k_s}$

$$W_o = d_i, 1 \tag{3}$$

where W_o denotes the output of system. $W_s = \in \bar{R}(\bar{W}s - 1 = [W_s, \ldots, Ws - 1]^{\mathrm{T}})$ denotes system input. $f_i(W_s)$ denotes the hysteresis type of nonlinearity. $F_{i,ji}(\bar{W}_s, ji)$ denotes the smooth function. For the proposed system, the state constraints are along with $-Q_{i,ji} < d_{i,ji} < \bar{Q}_{i,ji}$ denotes the positive design constraints, where $i = 1, \ldots, n, ji = 1, \ldots, s_i$. Considering the state constraints in our system results in stability of the system. We assume $\mu_i, \phi_i, \psi_i =$ design parameters where $\phi_i > 0$ are slopes of lines and $\phi_i > \psi_i$. It will exhibit the change in Eq. (1). Here, the states are modified after the constraints imposition as $f_i(w_s) = \phi_i(W_i(t))+ \rho_i(W_{s_i})$

$$d_{i,iz} = f_i(w_s) = \phi_i(W_i(t)) + \rho_i(W_{s_i}) + F_{i,z}(W_s, \bar{W}_s - 1) \tag{4}$$

where $\rho_i(Ws_i) = boundedandsatisfied$. All other parameters of system will remain same, and proposed system satisfies the state constraints for adaptive neural network used for translating a sentence from source to target language. The developed system exhibits BLEU, i.e. an automatic measure for translation accuracy as 61.02% on combining Keras model with bidirectional layer using gated recurrent units along with Relu and sigmoid activation function and then performing auto-tuning. Adam optimizer is also used to optimize the neural model.

3.4 SHH-MTS: hybrid approach

The hybrid approach uses extraction of linguistic feature and RNN to translate Sanskrit language to Hindi language. In this translation model, the source language is translated to the target language having a lexical gap. It is the process which is deep analysis of the source language and then its lexical transfer to the target language. Accuracy of hybrid approach is more than either only rule-based approach or only phrase-based approach. The SHH-MTS merges the best of both the approaches as it merges linguistic rich features with prominent deep learning approach to provide Sanskrit

to Hindi translation. The flow of this system is depicted in Fig. 5.

3.5 Deploying SHH-MTS on cloud

MTS is composed of multiple heterogeneous modules having dependencies according to the task performed. Resolving these complexities is not easy, and it is also a time-consuming task. The increasing demand in the request for MTS hinders the performance of the systems. It leads to slow response and requires more resources to provide more computation. This will increase the computational cost for most of the enterprises and academic institutions. Innovations are necessary to ride the inevitable tide of change. Recurrent changes in terms of either corpus, domain inclusion, the algorithm of modules, modifying rules or a combination of these to improve the accuracy, quality and performance of MT systems. Most of the developments are striving to reduce their computing cost through the means of virtualization. This demand of reducing the computing cost has led to the innovation of cloud computing (Qian et al. 2009). It offers better computing through improved utilization, reduced administration and infrastructure costs. It is a term used to describe both a platform and type of application. As a platform, it supplies, configures and reconfigures servers, while the servers can be physical machines or virtual machines. On the other hand, as applications that can be extended to be accessible through the internet and for this purpose large data centres and powerful servers are used to host the web applications and web services. Our proposed MTS is a web application hosted on cloud to provide SHH-MTS as a service.

The features of MTS deployed on the cloud are listed point-to-point below.

- It provides scalability at the same cost.
- Reduces load by the distribution of the task to different servers.
- Fast processing speed due to virtual machines
- Handling of frequent updates in the algorithm, rules, corpus, dictionary and domain inclusion.
- Enhancement, adaptability and scalability are easier to perform.
- Easy to use by the common user as no pre-requisite or knowledge of NLP is required.

These features are adopted by several researchers to develop MTS and by using MTS-as-a-service higher performance has been achieved. Nowadays different cloud service providers are assisting the industries as well as for personal use. Amazon Web Services (AWS) (Amazon 2020) is one of the cloud service providers, which is a secure cloud services platform offering compute power, database storage, content delivery and other functionality to help system scale-up and grow.

Algorithm 1: The Encoding of Input Sentence Algorithm

input : Parallel corpus P_c and Monolingual corpus M_c,$varaince = 0.01$, $mean = 0$,$V_\alpha = 0$
output: Context Vector s_i as summary of input sentence

1 *Encode* $W_s = w_{s_1}, w_{s_2}, w_{s_3}.....w_{s_z}$ *as* $s_i \in \mathbb{R}^{Ks}$;
// K_s is vocabulary size and z is input sentence length
2 *Encode* $W_h = w_{h_1}, w_{h_2}, w_{h_3}, ...w_{h_x}$ *as* $h_i \in \mathbb{R}^{k_h}$;
// K_h as vocabulary and x is the output sentence length
3 **for** $s = 1$, $s{+}{+}$, *while* $s < z$ **do**
4 *Tokenize the input sentence* $T(w_{s_1}, w_{s_2}, ...w_{s_n})$;
5 *Compute probability* $P_1(w_{s_1}, w_{s_2}....., w_{s_{t_z}}) = \prod_{i=1}^{t} P(w_{s_1}.....w_{s_{i-1}}) \approx \prod_{i=1}^{t} P(w_{s_i}|w_{s_1}....w_{s_{z-1}})....w_{s_{z-1}}$;
// probability of a sequence of token words conditioned on a window words rather than all previous words
6 *The data-set converted into sequence of integers-tokens*$(T(w(s_1)...w(s_(z))))$ *are then padded and truncated and saved as numpy arrays(np).*;
7 **for** *np.w(s)* **do**
8 *Apply Bi-directional Recurrent Neural Network(RNN)*;
9 **for** $<Ts = 1,\ Ts{+}{+},\ while\ Ts < Tz>$ **do**
10 *Forward RNN: Compute hidden state* $\vec{f} = (\vec{h_1}, \vec{h_2}, \vec{h_3}..\vec{h_{\tau_j}})$;
// read the sentence in forward order
11 **for** $Ts = z$, $Ts{-}{-}$, *while* $Tz < Ts$ **do**
12 *Backward RNN: Compute hidden state* $(\overleftarrow{h_1}, \overleftarrow{h_2}..., \overleftarrow{h_{\tau_j}})$;
// read the sentence in reverse order
13 *Compute annotation vector:* $H_i = [\vec{h_j}^T; \overleftarrow{h_j}^T]$.;
14 **for** *Each input word* s_z *map through hidden states* $\vec{h_\tau}$ *and* $\overleftarrow{h_\tau}$ **do**
15 *Compute embedding lookup:* $\vec{H_j} = f(\vec{h_{i-1}}, \bar{E}W_{s_n})$;
16 *The encoder computations are deeply stacked in following manner:*;
17 *Compute for first layer* **if** *i=1* **then**
18 $h_{t,1} = f_1(h_{t-1}, 1, w_s t)$ // $h_{t-1,i}$: previous time stamp value
19 **else** $h_{t,i} = f_{h_{t-1,i}, h_{t,i-1}}$ // $h_{t,i-1}$: previous layer in sequence value.
20 *Compute context vector* s_i *as summary of input sentence from Step 9 and 11*

Therefore, according to the need, the model can be accessed and utilized.

Virtualization (Xing and Zhan 2012) is a viable option, by making an application function as a repository. The key benefits of developing virtual appliance are fine granularity with reducing time for adding and removing computational resources. It would also increase the mobility of application and reduce deployment time. The deployment of virtualization can be performed on the cloud as well as stand-alone. A SHH-MTS is deployed on the cloud architecture as in Fig. 6. The proposed architecture is divided into three layers, i.e. interface layer, service layer and database layer. The interface

Algorithm 2: Generation of next hidden state using Grated Recurrent Unit(GRU) Algorithm

input : Previous state h_{t-1} and input w_{s_t}
output: Next hidden state h_t

Forward States: Bi-Directional Recurrent Neural Network;
// Grated Recurrent Units(GRU) is designed in a manner to have more persistent memory thereby making it easier for RNN to capture long term dependency.
foreach *s(i)* **do**
  for *i=1,i>z,i++* **do**
    $\vec{up}_i = \sigma(\vec{W}_{up}\bar{E}_{s_i} + \vec{O}_{up}\vec{h_{i-1}})$;
    // $d = dimensionality of word embedding$ and u is number of hidden units $\bar{E} \in \mathbb{R}^{d \times k_s}$
    $\vec{res}_i = \sigma(\vec{W}_{res}\bar{E}s_i + \vec{O}_{res}\vec{h_{i-1}})$;
    // $\vec{W}, \vec{W}_{up}, \vec{W}_{res} \in \mathbb{R}^{u \times d}$
    $\vec{h}_i = tanh(\vec{W}\bar{E}s_i + \vec{O}[\vec{res}_i \odot \vec{h_{i-1}}]$;
    // $\vec{O}, \vec{O}_{up}, \vec{O}_{res} \in \mathbb{R}^{u \times u}$
    $h_i = (1 - \vec{up}_i) \odot \vec{h_{i-1}} + up_i \odot \vec{h}_i)$;
    // σ is logistic sigmoid function
  Backward states of bidirectional recurrent Neural network
  for *i=1,i>z,i++* **do**
    $\overleftarrow{up}_i = \sigma(\overleftarrow{W}_{up}\bar{E}_{s_i} + \overleftarrow{O}_{up}\overleftarrow{h_{i-1}})$;
    $\overleftarrow{res}_i = \sigma(\overleftarrow{W}_{res}\bar{E}s_i + \overleftarrow{O}_{res}\overleftarrow{h_{i-1}})$;
    $\overleftarrow{h}_i = tanh(\overleftarrow{W}\bar{E}s_i + \overleftarrow{O}[\overleftarrow{res}_i \odot h - \overleftarrow{i} - 1]$;
    $h_i = (1 - \overleftarrow{up}_i) \odot \overleftarrow{h_{i-1}} + up_i \odot \overleftarrow{h}_i)$;
    $h_i = [\vec{h}_i + \overleftarrow{h}_i]$ // Combining forward and backward states

Algorithm 3: Embedding of linguistic features extracted from pipeline rule-based architecture algorithm

input : Each feature with distinct word embedding $s_z y$ and
output: Context Vector s_i as summary of input sentence along with linguistic features

Combining all these word vectors $s_z y$ form an feature embedding matrix $E \in \mathbb{R}^{dy \times ky}$;
// dk is summation of dimension of all feature embedding and ky as vocabulary size of K^{th} feature
These embedding are concatenated with total embedding size as the length matches. The input embedded sentence vectors are multiplied with these extracted linguistic features.;
// K_h as vocabulary and x is the output sentence length
for $s_z y = 1$, $s_z y{+}{+}$, *while* $s_{zy} < z$ **do**
  $h_l = tanh(\vec{W}\prod_y^F \bar{E}_y s_{zy} + \vec{O}\vec{h}_{l-1})$;
  for *i=1,i>zy,i++* **do**
    Calculates association between input word W_s to produce the next output wordw_h by calculating the impact of word representation($\vec{h}_i, \overleftarrow{h}_i$) $h_j = (\vec{h}_i, \overleftarrow{h}_i)$;
for *i=1,i>z,i++* **do**
  for *j=1,j>z,j++* **do**
    Calculate alignment model a_{ij} // Output position around i to input position around j.
    Hidden state d_{i-1} and h_j // j^{th} annotation of input Sanskrit sentence.
    $a_{ij} = J_a^{\tau} tanh(W_a d_{i-1} + O_a h_j)$
    // $W_a \in \mathbb{R}^{n'_1}$, $O_a \in \mathbb{R}^{n' \times n}$, $J_a \in \mathbb{R}_{n' \times 2n}$ are weight matrices
    $\alpha_{ij} = \frac{exp(a_{ij})}{\sum_{y=1}^{Ts} exp(a_{iy})}$ $S_i = \sum_{j=1}^{ts} \alpha_{ij} h_j$
    // $S = feed - forward Neural Network$
    // The computed scalar attention value is normalized using softmax activation function, so all input words s adds up to 1

layer is used for user interaction. In the proposed architecture, the interface is built in the form of a website for SHH-MTS, delivering Sanskrit–Hindi translation as a service to the users.

All the users requesting for Sanskrit–Hindi translation can access the system through the interface of the website which is deployed at the service layer. At the back end of the translation system, the output is generated by various linguistic tools and encoder–decoder architecture of RNN. All these user requests are stored in the cloud storage repository and passed further for processing into the SHH-MTS. Even load balancing and auto-scaling are performed on this layer to handle the traffic demands. It also manages to address advance routing needs by dynamically scaling the web application to changing traffic on demand. It can create capacity groups of servers that work accordingly to the demands. The AWS Elastic Compute Cloud (EC2) allows users to use virtual machines of different configurations as per requirements. It provides a more secure model for every host. The database layer is comprised of the parallel corpus, monolingual corpus and dictionaries. It is used for the physical interface between the application and the database. Simple storage service (S3) is used to allow users to store and retrieve various types of data API calls. With provisioning of additional hardware resources, it is possible to keep the response time within optimum limits as the load increases but this increases cost.

So, the objective of this proposed work is to deploy MTS, on the cloud with provisioning of larger computation resources. It will help to increase the scalability of the system and to improve response time. Cloud deployment requires optimum resource utilization which is possible only when an application can scale-up and scale-down rapidly. This is easily feasible for our proposed SHH-MTS to scale-up or scale-down in real-time. In this work, a hybrid MTS model is deployed on cloud of Amazon (EC2) as depicted in Fig. 7 having better accuracy, CPU utilization and minimum response time. It also eases the deployment and scope of extension or manageability due to which the performance of our proposed system is better than our stand-alone version of systems.

Algorithm 4: Decoding the target sentence from context vector of linguistic features extracted from pipeline rule-based architecture

input : Previous hidden state d_{i-1}, some representation of input context s_i and embedding of previous word output $E_{h_{i-1}}$
output: New output decoder hidden stated_i

for *i=1,i>z,i++* **do**
  if *d=o* **then**
    $d_0 = f(W_d \vec{h_1})$
  // For initial hidden state $W_d \in \mathbb{R}^{d \times d}$.
  else $\bar{d_i} =$
    $tanh(WEh_{y_{i-1}}) + O[res_i + d_{i-1}] + Ss_i)$// hidden state d_i is computed given annotation from encoder and for update and Reset
  $up_i = \sigma(W_{up} E_{h_i - 1} + O_{up} d_{i-1} S_{up} s_i)$;
  $res_i = \sigma(W_{res} Eh_{i-1} + O_{res} h_{i-1} + S_{res} S_i)$;
  // E= Embedded matrix of word for target language, u=Number of hidden units,d=word embedding dimension. $W, W_{up}, W_{res} \in \mathbb{R}^{u \times d}$
  // 0, $O_{up}, O_{res} \in \mathbb{R}^{d \times 2d}$ are weight matrices.

Algorithm 5: Decoding the target sentence from context vector of linguistic features extracted from pipeline rule-based architecture

input : Decoder hidden state d_{i-1}, input context s_i and embedding of previous output word h_{i-1}
output: New word prediction w_{h_i}

The vector for prediction p_i for a output word;
for *i=1,i>z,i++* **do**
  for *j=1,j>l,j++* **do**
    $p_i = [max \bar{p}_i, 2j-1, \bar{i}, 2j]^{\tau}_{j=1\ldots,l}$ // Repeat $E_{W_{h_i - 1}}$ for d_{i-1} rather than d_i as it fragments the encoder state progress from d_{i-1} to d_i for prediction of output word p_i
    foreach *Output Word wh* **do**
      Calculate w_{h_i} as highest value in the vector
        $p_i = [max \bar{p}_i, 2j-1, \bar{i}, 2j]^{\tau}_{j=1\ldots,l}$
        // Training is performed accordingly as network being aware of correct output w_{h_i} assigns larger probability value
    $prob(h_i | d_{i-1}, s_i) \propto (h^{\tau} W_o pi)$ // Activation function softmax is used to convert raw vector into a probability distribution having sum of values as 1
    // ReLu combines input to yield the next hidden state

4 Experimental details

The experiment has been conducted on developed SHH-MTS to measure various QoS parameters. The technical specifications along with its versions are displayed in Table 4. For the deployment of SHH-MTS on cloud infrastructure, Amazon Elastic Computer Cloud (EC2) is used. It stimulates scalable deployment of MTS by providing a web service. It provides secure, resizable compute capacity through which we have booted an Amazon Machine Image (AMI) 3.0 for deep learning to configure a virtual machine (VM), also called an instance. The developed system includes (dictionaries, parallel corpus, monolingual corpus, program codes, algorithms, lexical resources, rule database, machine-learned data and its models) is packed as AMI. The experiment conducted was based on Linux Kernel version 3.4.34 Operating system through which user program interacts with the kernel. Virtualization was first performed with Xen and later with Compute Optimized C5 instance, which was based on custom architecture around kernel-based virtual machine (KVM) hypervisor. The processor was ported with IA-64. Elastic Block Store (EBS) volume as a rooted device was used for storage with G3.4*large type. It provides raw block devices that can be attached to the Amazon EC2 instance. It also supports cloning and snapshot, so we have cloned the system image to other virtual machines. It is built on replicating storage when the failure of one component would not cause data loss. EBS volumes can be attached or detached from instance or VM, while they are running and moved from one VM to another VM. A simple storage (S3) is also used which we access (read and write) through the API. The experiment was based on 16 core processor and 122 GB RAM. Rule-based auto-scaling was used which adapts according to the CPU utilization threshold though it takes several seconds to scale up and scale down. The VM start-up time is not dependent on VM type, AMI size, data centre location, etc. Still it took a few seconds to configure.

Each virtual machine has a GPU with 122 GB of RAM and 16-core processor to achieve a high throughput speed approximately 2500 words per second. This speed is not possible for normal systems because in this one epoch will take approximately 2 h to run. We have allocated 5 nodes for this experiment. On each node, SHH-MTS is pre-installed with the help of AMI of 1GB size. It takes 60 s to boot. For each instance created, we have 12 elastic computing unit and 4 virtual CPUs and a memory of 61 GB for a single instance (EBS storage only) with high network performance. After all the environment set-up, the system is ready to run and perform performance testing. The system first runs rule-based linguistic tools, which gives its output to CSV file. This file is processed on TensorFlow with Keras with python 3. It has been evaluated on parameters such as average response time, cost optimization, throughput, server load, total time taken concerning rule matching probability. The number of computational resources (Storage, O.S, Instance type), data, virtual machines were varied across to evaluate the performance of the MTS. The interface designed and hosted on the cloud is shown in Fig. 8.

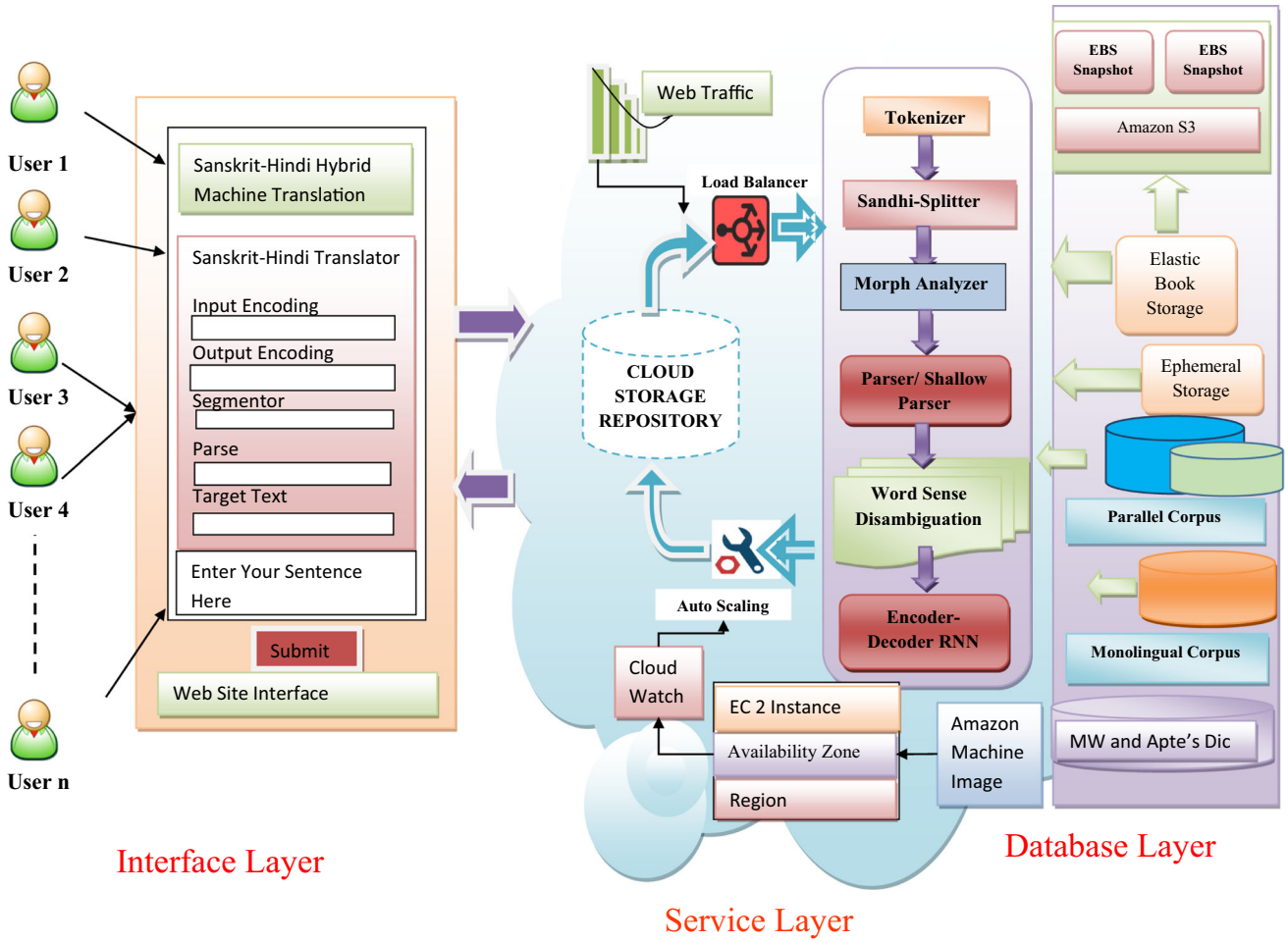

Fig. 6 Architecture for proposed system

Table 4 Technical specifications along with versions

Technical specifications	Version/number
Amazon machine image (AMI)	3.0
Linux kernel O.S	3.4.34
Compute-intensive virtual machine	C5
Elastic block storage (EBS)	G3.4*large type
16-core processor	IA-64
RAM	122 GB
Virtual machines	(4, 8, 12, 16)
GPU	Geforce GTX 980
Memory (for each instance)	61 GB
Elastic compute unit	12
CPU	2 (for each VM)
Cache	4 MB

5 Result analysis

The performance analysis of this proposed work was conducted by performing a diverse statistical test. The entire section is been classified into two subsections. The first section consists of a comparative analysis of our proposed SHH-MTS as a service with the existing works, while the later subsection focuses on illustrating the performance of MTS based on different approaches rule-based, neural-based and hybrid. As the proposed system was built in iterations, i.e. first rule-based approach was applied. Later neural-based and at last, the output of rule-based was feed as features to neural-based forming it as a hybrid approach. The different statistical tests were conducted for evaluating the performance such as average response time to rule-matching probability, auto-tuning process, cost optimization, throughput, performance, server load and total time is taken to the number of virtual machines. For each of the statistical test conducted, results were achieved by implementing multiple runs which manifested multiple units of values at different time instances.

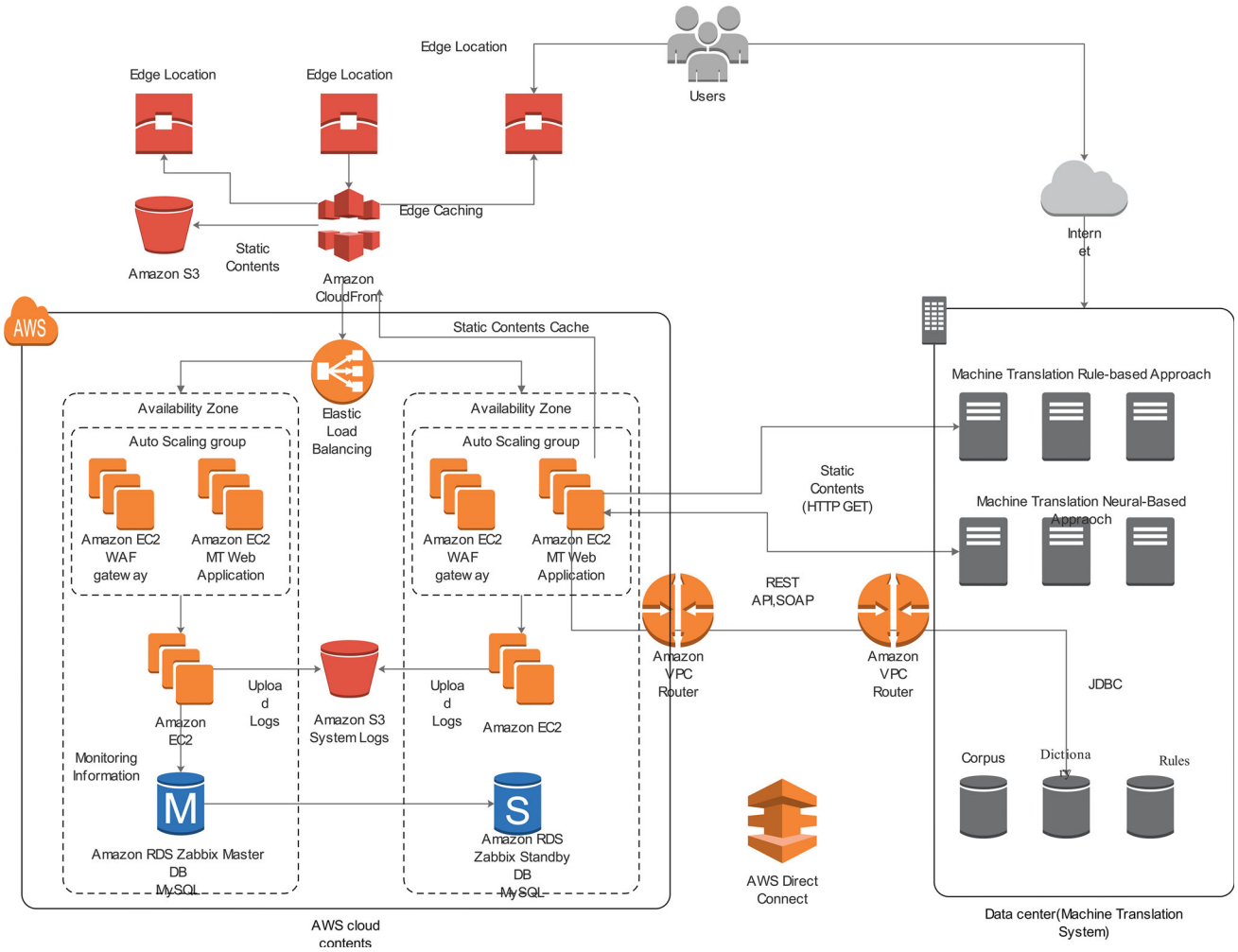

Fig. 7 AWS infrastructure used for deploying SHH-MTS as a service

5.1 Comparative analysis of our proposed work with earlier research work

Throughput is directly proportional to the number of processes completed, and it is also used to analyse the performance of the translation system. It is calculated based on resources used and time consumed.

- Throughput on stand-alone MTS: Its calculation is done by executing a whole book of sentences using 16 core processor, 122 GB RAM, EBS only instances and G3.4x large type. The same job is divided into several tasks to execute on different computing resources. Table 5 highlights the results of the stand-alone system of earlier proposed work (Ahmad et al. 2011a) on Hindi to Punjabi Cloud-based Sampark MTS tested on dual-core CPU and 1 GB of RAM and our proposed SHH-MTS as a service.
- Throughput on virtual machine: at first we allocated 2 virtual machine which took 5440 s for processing, later we experimented with 4 virtual machine which took 2113 s, this time for processing reduced rapidly on using more number of virtual machine as it took 790 s for 8 virtual machines and 320 s for 12 virtual machines. Comparing with CBSMTS, it took 970 s on 12 virtual machines as depicted in Table 6.
- Computing nodes reducing the time for computation: As depicted in Figs. 9 and 10, we performed the test for different sentence lengths, i.e. 20, 50, 100, 150, 200 and 250 and for varying computing nodes 2, 3, 4, 5, 6 and 7. The results can be seen that on increasing the nodes the time is decreasing.
- Response time: The response time is the time required for output sentence after providing the input sentence to MTS. It can be seen from the results in Fig. 11 the time taken for our proposed CBSHH-MTS and previous research work for different sentence lengths, i.e. 20, 50, 100, 150, 200, 250.

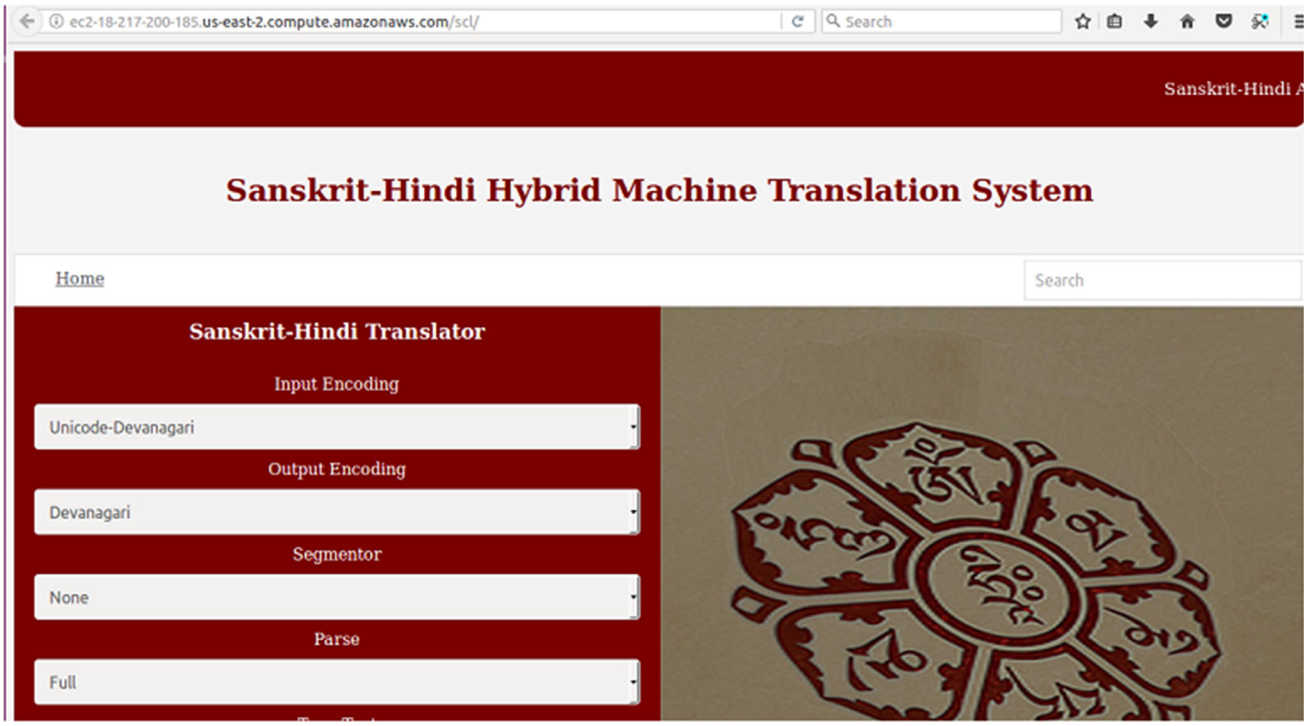

Fig. 8 SHH-MTS as a service

Table 5 Throughput results on stand-alone

Book	Total time taken on stand-alone
Nirmal by Kumar et al. (2013a)	198 min + 13 s
Sankshepa RaamaayaNam (on SHH-MTS)	130 min + 50 s

Table 6 Throughput calculation on virtual machines

Number of VM	Throughput on CBSHH-MTS	Throughput on Kumar et al. (2013a)
2	90 min + 40 s	112 min + 19 s
4	35 min + 60 s	54 min + 6 s
8	13 min + 13 s	26 min + 26 s
12	5 min + 20 s	16 min + 10 s

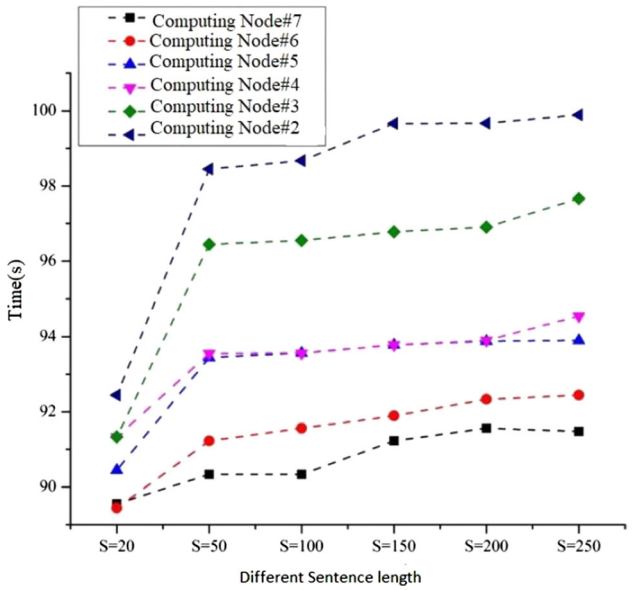

Fig. 9 Deployment and usage of cloud infrastructure for (Kumar et al. 2013a)

5.2 Performance analysis on deployment of rule-based MT, neural MT and hybrid MT on cloud

The elaborate description of rule based used in the development of our proposed system is given in Sect. 3.1, neural based in Sect. 3.2 and hybrid in Sect. 3.3. These have been deployed on cloud individually, and their performance is analysed based on various parameters. The average response time which would depend on the matching of rules from the rule database is depicted in Fig. 12, while the average response time on the number of match action rules is depicted in Fig. 13. The CPU utilization on cloud-based on the packet arrival rate is shown in Fig. 14. Though the AWS provides pay as you use service, the cost according to resources required is displayed in Fig. 15. The throughput which is directly proportional to the number of processes completed and calculated based on resources used and time consumed in Fig. 16. The

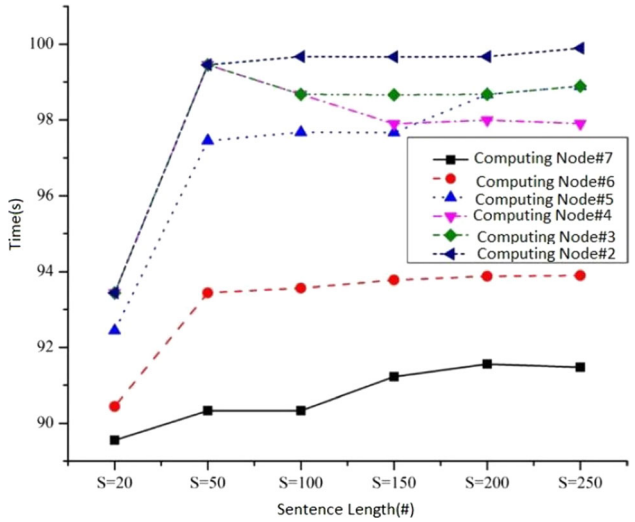

Fig. 10 Deployment and usage of cloud infrastructure for SHH-MTS as a service

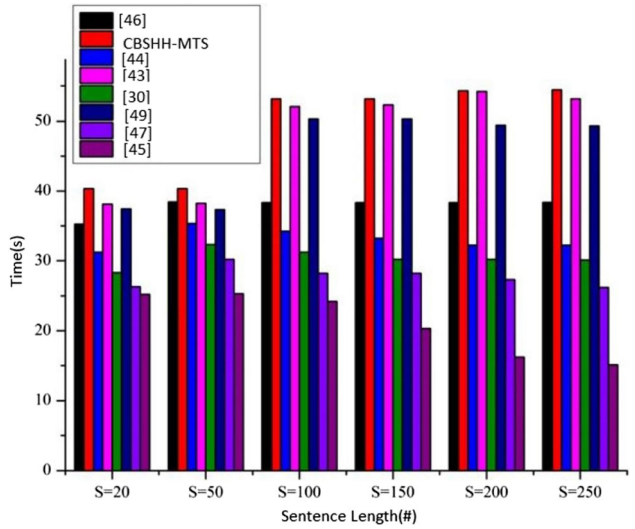

Fig. 11 Comparison of SHH-MTS as a service with the existing work based on the time consumed for different sentence lengths

throughput will increase by increasing the number of virtual machines as in Fig. 17. The server load will decrease in hybrid approach as in Fig. 18 as compared to rule based and neural based. The auto-tuning process (addition of hidden layers automatically) which reduced significantly for both systems after the deployment of the proposed system on the cloud. The graphs depict SHH-MTS, i.e. hybrid performs better than rule-based approach and neural-based approach.

6 Open challenges and future research directions

Based on the literature survey, following challenges have been recognized and listed for the future research in this area.

1. QoS enhancement of MTS for the end-users needs to be provided (Dastjerdi et al. 2011). The three dimensions of QoS (job completion time, system throughput, system performance) can be applied to other applications of NLP that exhibits list homomorphism behaviour and applications which can be partitioned for distributed processing like cloud computing (Kumar et al. 2013a).
2. Evaluation of cloud-based models for huge data sets needs to be performed. A comparison of various cloud-based language models based on various parameters is also desirable (Ferrández-Tordera et al. 2016).
3. Developing an MTS as the virtual appliance can handle the most common problem of MTS, i.e. updates by making it function as a repository. The key benefit of developing virtual appliance is the fine granularity with reducing time for adding and removing computational resources. Virtualization also increases the mobility of application and reduces deployment time. The virtualization can be deployed both on the cloud and stand-alone systems (Kumar et al. 2013c).
4. Cloud-based deployment can be performed for various other NLP applications such as text to speech, automatic speech recognition and transliterated system for enhanced performance (Ahmad et al. 2011a).
5. The experiment performed on deploying MT on cloud needs to be performed on public cloud service providers with symmetric computing resources in cloud environment (Ahmad et al. 2011a).
6. There is a need for a special version of the platform for compute-intensive NLP application. As such applications which require more time in computing rather than processing data, cloud platform needs to adopt such applications and develop a special version that can be used by others with an ease (Ahmad et al. 2011a).
7. The scalable model-parallelism library available for training giant neural networks (Huang et al. 2019) needs to decrease the re-computation time, load imbalance, reducing the scaling time for encoder and decoder (Vaswani et al. 2018), parallelism at word level processing (Ahmad et al. 2014) and also minimizes the performance overhead.

7 Conclusion

SHH-MTS is the integration of linguistically rich approach, i.e. rule-based with prominent result-oriented approach, i.e. neural-based which is gaining significant attention nowadays. It is a complex application with a large number of heterogeneous modules. Deploying such a complex application on a stand-alone machine becomes a difficult and time-consuming task. The existing Sanskrit–Hindi MTS has

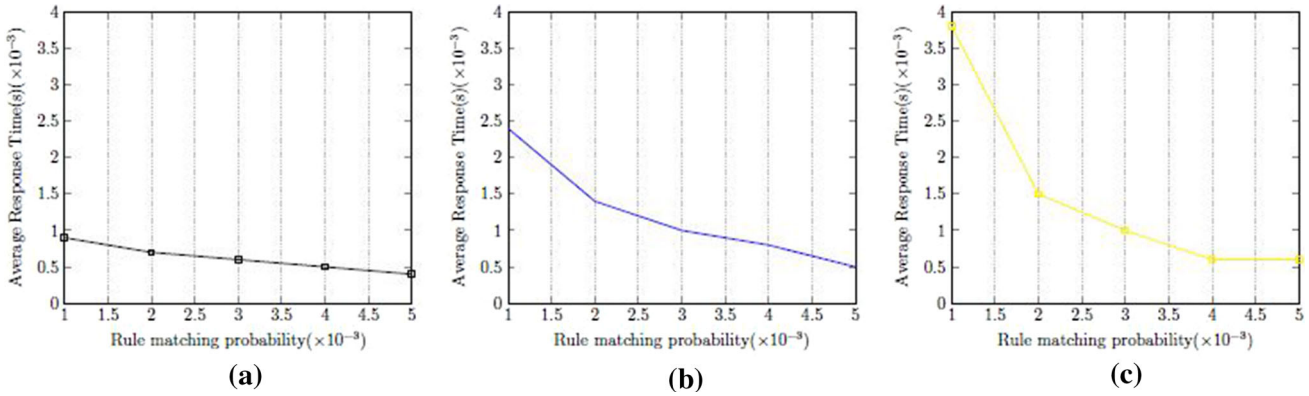

Fig. 12 Average response time pertaining to rule matching probability, **a** hybrid MTS, **b** neural-based MTS and **c** rule-based MTS

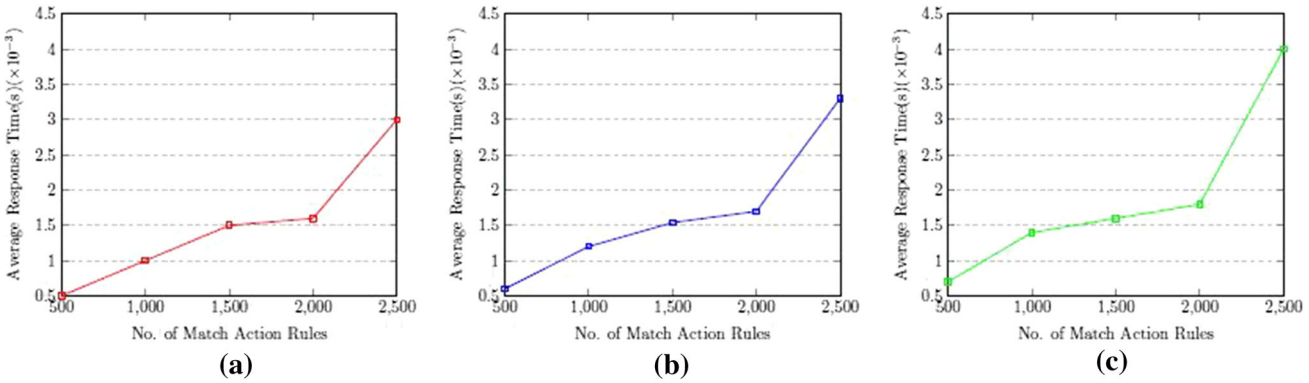

Fig. 13 Average response time pertaining to number of matching action rules, **a** hybrid MTS, **b** neural-based MTS and **c** rule-based MTS

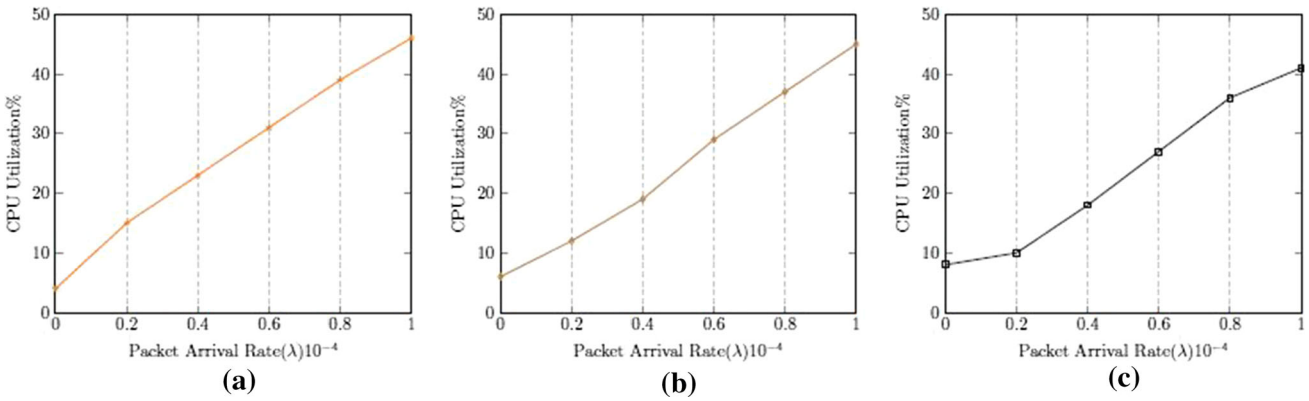

Fig. 14 CPU utilization pertaining to packet arrival rate, **a** hybrid MTS, **b** neural-based MTS and **c** rule-based MTS

many drawbacks such as slow speed, less data accuracy and low response time. All these factors adversely affect the performance of the system. It has been observed that local server takes more time to respond and provides lesser accuracy. Therefore, offering MTS as a cloud service is a better proposition for increasing its performance in terms of accuracy and response time. Moreover, auto-tuning for neural-based MT is not possible at the local host due to memory issues, but it is possible on the cloud. Several layers are added automatically to attain maximum accuracy and high speed. The proposed CBSHH-MTS provides better throughput, rule matching probability and number of matching rules in comparison to the stand-alone systems.

8 Future work

In future, massive data can be accessed using the multilingual platform, and the cloud can support this environment too. It can also be released as a virtual appliance repository,

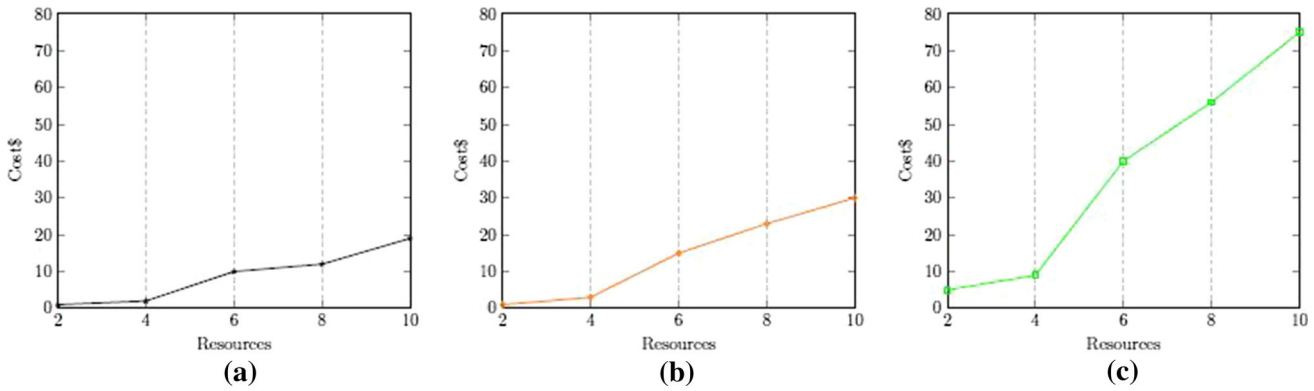

Fig. 15 Cost pertaining to resources, **a** hybrid MTS, **b** neural-based MTS and **c** rule-based MTS

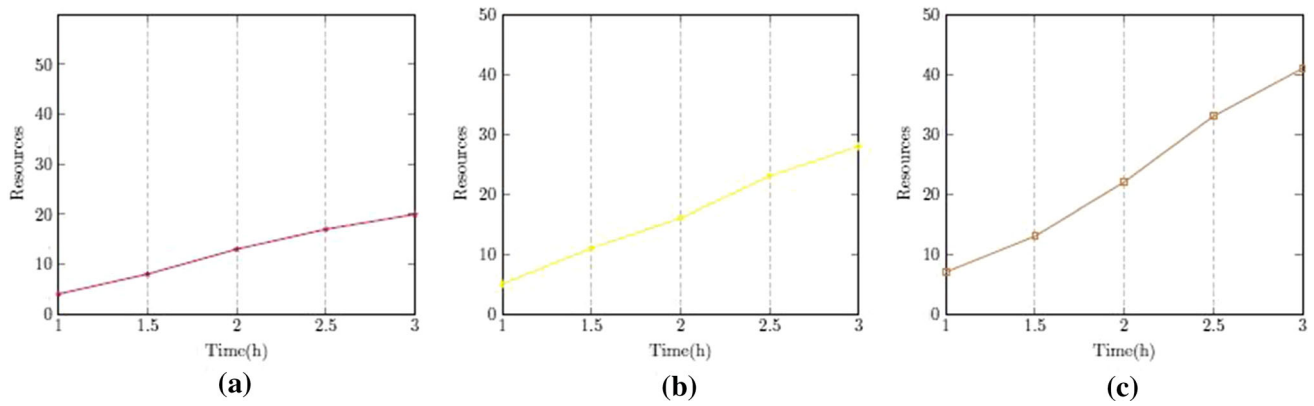

Fig. 16 Throughput, **a** hybrid MTS, **b** neural-based MTS and **c** rule-based MTS

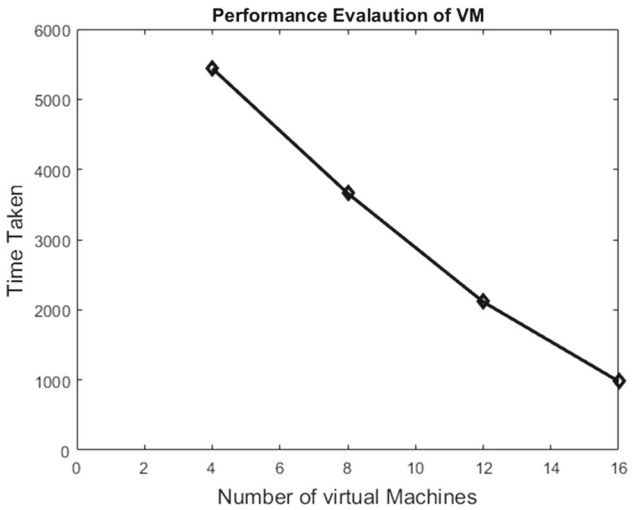

Fig. 17 Time taken with respect to number of virtual machines

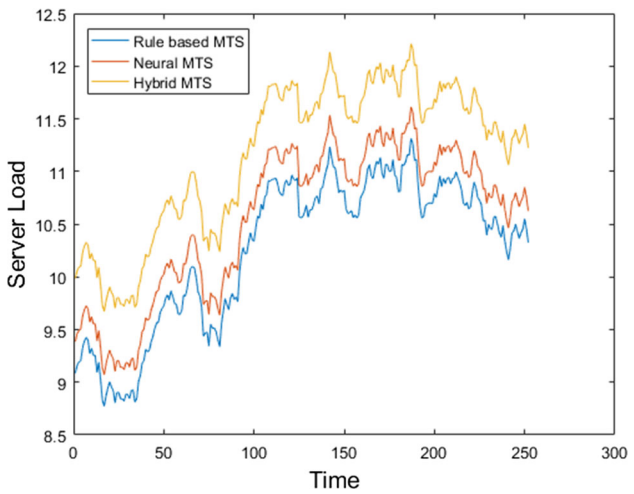

Fig. 18 Server load with respect to time

which can be downloaded easily and used. It will handle the most common problem of MTS, i.e. frequent updates. Also, we would extend our work by using the aforementioned one-to-one nonlinear mapping, the strict-feedback system with full-state constraints which will be converted into a pure-feedback system without state constraints. This will significantly reduce the computational complexity, and stability of the nonlinear system will be enhanced (Sun et al. 2018).

Compliance with ethical standards

Conflict of interest All the authors declare that there is no conflict of interest.

Human or animal participants This article does not contain any studies with human or animal participants performed by any of the authors.

References

Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M et al (2016) Tensorflow: a system for large-scale machine learning. OSDI 16:265–283

Abd-El-Malek M, Wachs M, Cipar J, Sanghi K, Ganger GR, Gibson GA, Reiter MK (2012) File system virtual appliances: portable file system implementations. ACM Trans Storage (TOS) 8(3):9

Ahmad R (2013) Engineering machine translation for deployment on cloud. PhD thesis. International Institute of Information Technology Hyderabad, India

Ahmad I, Ranka S (2016) Handbook of energy-aware and green computing-two volume set. CRC Press, Boca Raton

Ahmad R, Kumar P, Rambabu B, Sajja P, Sinha MK, Sangal R (2011a) Enhancing throughput of a machine translation system using mapreduce framework: an engineering approach. In: ICON

Ahmad R, Rathaur A, Rambabu B, Kumar P, Sinha MK, Sangal R (2011b) Provision of a cache by a system integration and deployment platform to enhance the performance of compute-intensive NLP applications. In: African conference on software engineering applied computing

Ahmad R, Kumar P, Kumar A, Sinha MK, Chaudhary B (2014) Improve user experience on web for machine translation system using storm. In: IEEE 4th international conference on big data and cloud computing. IEEE, New York, pp 243–248

Amazon E (2020) Amazon web services. Accessed on 28 Mar 2020

Asemi A, Salim SSB, Shahamiri SR, Asemi A, Houshangi N (2019) Adaptive neuro-fuzzy inference system for evaluating dysarthric automatic speech recognition (ASR) systems: a case study on MVML-based ASR. Soft Comput 23(10):3529–3544

Bahadur P, Jain A, Chauhan D (2012) EtranS-A complete framework for English to Sanskrit machine translation. In: International journal of advanced computer science and applications (IJACSA) from international conference and workshop on emerging trends in technology. Citeseerx, New York

Bahdanau D, Cho K, Bengio Y (2014) Neural machine translation by jointly learning to align and translate. Preprint arXiv:1409.0473

Barkade V, Devale PR (2010) English to Sanskrit machine translation semantic mapper. Int J Eng Sci Technol 2(10):5313–5318

Bharati A, Kulkarni A (2007) Sanskrit and computational linguistics. In: 1st international Sanskrit computational symposium. Department of Sanskrit Studies, University of Hyderabad

Bharati A, Chaitanya V, Sangal R, Ramakrishnamacharyulu K (1995) Natural language processing: a paninian perspective. Prentice-Hall, New Delhi

Bharati A, Kulkarni AP, Sheeba V (2006) Building a wide coverage Sanskrit morphological analyser: a practical approach. In: The 1st national symposium on modelling and shallow parsing of Indian Languages. IIT, Bombay

Chaudhury S, Rao A, Sharma DM (2010) Anusaaraka: an expert system based machine translation system. In: Proceedings of the 6th international conference on natural language processing and knowledge engineering (NLPKE-2010). IEEE, New York, pp 1–6

Chen Y, He J, Zhang X, Hao C, Chen D (2019) Cloud-DNN: an open framework for mapping DNN models to cloud FPGAs. In: Proceedings of the 2019 ACM/SIGDA international symposium on field-programmable gate arrays, pp 73–82

Cho K, Van Merriënboer B, Bahdanau D, Bengio Y (2014) On the properties of neural machine translation: encoder–decoder approaches. Preprint arXiv:1409.1259

Chowdhury GG (2003) Natural language processing. Ann Rev Inf Sci Technol 37(1):51–89

Dastjerdi AV, Garg SK, Buyya R (2011) Qos-aware deployment of network of virtual appliances across multiple clouds. In: IEEE 3rd international conference on cloud computing technology and science. IEEE, New York, pp 415–423

Dong Y, Zhang Z, Hong WC (2018) A hybrid seasonal mechanism with a chaotic cuckoo search algorithm with a support vector regression model for electric load forecasting. Energies 11(4):1009

Dunn NA, Unni DR, Diesh C, Munoz-Torres M, Harris NL, Yao E, Rasche H, Holmes IH, Elsik CG, Lewis SE (2019) Apollo: democratizing genome annotation. PLoS Comput Biol 15(2):e1006790

Ferrández-Tordera J, Ortiz-Rojas S, Toral A (2016) Cloudlm: a cloud-based language model for machine translation. Prague Bull Math Linguist 105(1):51–61

Foster I, Zhao Y, Raicu I, Lu S (2008) Cloud computing and grid computing 360-degree compared. In: Grid computing environments workshop. IEEE, New York, pp 1–10

Fox A, Griffith R, Joseph A, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I (2009) Above the clouds: a Berkeley view of cloud computing. Dept Electrical Eng and Comput Sciences, University of California, Berkeley, Rep UCB/EECS 28(13):2009

Galaviz-Aguilar JA, Roblin P, Cárdenas-Valdez JR, Emigdio Z, Trujillo L, Nuñez-Pérez JC, Schütze O et al (2019) Comparison of a genetic programming approach with anfis for power amplifier behavioral modeling and FPGA implementation. Soft Comput 23(7):2463–2481

Gao Q, Vogel S (2010) Training phrase-based machine translation models on the cloud: open source machine translation toolkit chaski. Prague Bull Math Linguist 93:37–46

Gorzalczany MB, Gluszek A (2002) Neuro-fuzzy systems for rule-based modelling of dynamic processes. In: Zimmermann HJ, Tselentis G, van Someren M, Dounias G (eds) Advances in computational intelligence and learning. Springer, Dordrecht, pp 135–146

Goyal P, Arora V, Behera L (2009) Analysis of Sanskrit text: parsing and semantic relations. In: Huet G, Kulkarni A, Scharf P (eds) Sanskrit computational linguistics. Springer, Berlin, Heidelberg, pp 200–218

Hellwig O (2009) Sanskrittagger: a stochastic lexical and pos tagger for Sanskrit. In: Huet G, Kulkarni A, Scharf P (eds) Sanskrit computational linguistics. Springer, Berlin, Heidelberg, pp 266–277

Hellwig O (2010) Performance of a lexical and pos tagger for Sanskrit. In: Jha GN (ed) Sanskrit computational linguistics. Springer, Berlin, Heidelberg, pp 162–172

Hong WC, Li MW, Geng J, Zhang Y (2019) Novel chaotic bat algorithm for forecasting complex motion of floating platforms. Appl Math Model 72:425–443

Huang Y, Cheng Y, Bapna A, Firat O, Chen D, Chen M, Lee H, Ngiam J, Le QV, Wu Y (2019) Gpipe: efficient training of giant neural networks using pipeline parallelism. In: 33rd conference on neural information processing systems (NeurIPS 2019). Advances in Neural Information Processing Systems, Vancouver, Canada, pp 103–112

Huet G (2006) Shallow syntax analysis in Sanskrit guided by semantic nets constraints. In: Proceedings of the 2006 international workshop on research issues in digital libraries. ACM, New York, pp 1–10

Jha GN, Agrawal M, Mishra SK, Mani D, Mishra D, Bhadra M, Singh SK et al (2009) Inflectional morphology analyzer for Sanskrit. In: Huet G, Kulkarni A, Scharf P (eds) Sanskrit computational linguistics. Springer, Berlin, Heidelberg, pp 219–238

Ketkar N (2017) Introduction to Keras. In: Deep Learning with Python. Apress, Berkeley, CA, pp 97–111
Kim HN, Yoo SH, Kim KH, Chung AYJ, Lee JY, Lee SK, Jung JT (2019) Method for controlling hand-over in drone network. US Patent 10,230,450
Kiyurkchiev V, Pavlov N, Rahnev A (2019) Cloud-based architecture of dispel. Int J Pure Appl Math 120(4):573–581
Kulkarni A (2013) A deterministic dependency parser with dynamic programming for Sanskrit. In: Proceedings of the 2nd international conference on dependency linguistics (DepLing 2013), pp 157–166
Kulkarni A, Kumar A (2011) Statistical constituency parser for Sanskrit compounds. In: Proceedings of ICON
Kulkarni A, Ramakrishnamacharyulu K (2013) Parsing Sanskrit texts: some relation specific issues. In: Proceedings of the 5th international Sanskrit computational linguistics symposium. DK Printworld (P) Ltd
Kulkarni A, Pokar S, Shukl D (2010) Designing a constraint based parser for Sanskrit. In: Jha GN (ed) Sanskrit computational linguistics. Springer, Berlin, Heidelberg, pp 70–90
Kumar A, Sheebasudheer V, Kulkarni A (2009) Sanskrit compound paraphrase generator. In: Proceedings of ICON
Kumar A, Mittal V, Kulkarni A (2010) Sanskrit compound processor. In: Jha GN (ed) Sanskrit computational linguistics. Springer, Berlin, Heidelberg, pp 57–69
Kumar P, Ahmad R, Chaudhary B, Sinha M (2013a) An approach to assure QoS of machine translation system on cloud. In: Proceedings of the 4th international conference on cloud computing, GRIDs, and virtualization, pp 179–184
Kumar P, Ahmad R, Chaudhary B, Sinha M (2013b) Dashboard: a tool for integration, validation, and visualization of distributed NLP systems on heterogeneous platforms. In: The companion volume of the proceedings of IJCNLP 2013: system demonstrations, pp 9–12
Kumar P, Ahmad R, Chaudhary BD, Sangal R (2013c) Machine translation system as virtual appliance: for scalable service deployment on cloud. In: IEEE 7th international symposium on service-oriented system engineering. IEEE, New York, pp 304–308
Kundra H, Sadawarti H (2015) Hybrid algorithm of cuckoo search and particle swarm optimization for natural terrain feature extraction. Res J Inf Technol 7(1):58–69
Laadan O, Nieh J, Viennot N (2010) Teaching operating systems using virtual appliances and distributed version control. In: Proceedings of the 41st ACM technical symposium on computer science education. ACM, New York, pp 480–484
Machida F, Kawato M, Maeno Y (2010) Renovating legacy distributed systems using virtual appliance with dependency graph. In: 2010 international conference on network and service management. IEEE, New York, pp 17–24
Mishra V, Mishra R (2009) Ann and rule based model for English to Sanskrit machine translation. INFOCOMP J Comput Sci 9(1):80–89
Mittal V (2010) Automatic Sanskrit segmentizer using finite state transducers. In: Proceedings of the ACL 2010 student research workshop, association for computational linguistics, pp 85–90
Naderpour H, Mirrashid M (2019) Classification of failure modes in ductile and non-ductile concrete joints. Eng Fail Anal 103:361–375
Naderpour H, Mirrashid M (2020a) Confinement coefficient predictive modeling of FRP-confined RC columns. Adv Civ Eng Mater 9(1):1–21
Naderpour H, Mirrashid M (2020b) Proposed soft computing models for moment capacity prediction of reinforced concrete columns. Soft Comput 2020:1–15
Naderpour H, Vahdani R, Mirrashid M (2018) Soft computing research in structural control by mass damper (a review paper). In: 4th international conference on structural engineering, Tehran
Naderpour H, Mirrashid M, Nagai K (2019) An innovative approach for bond strength modeling in frp strip-to-concrete joints using adaptive neuro-fuzzy inference system. Eng Comput 2019:1–18
Nandi M, Ramasree R (2013) Rule-based extraction of multi-word expressions for elementary Sanskrit texts. Int J Adv Res Comput Sci Softw Eng 3(11):661–667
Neelaveni R, Sridevi B (2019) A novel Neyman–Pearson criterion-based adaptive neuro-fuzzy inference system (NPC-ANFIS) model for security threats detection in cognitive radio networks. Soft Comput 23(18):8389–8397
Pandey RK, Jha GN (2016) Error analysis of sahit—a statistical Sanskrit–Hindi translator. Proc Comput Sci 96:495–501
Pappu A, Sanyal R (2008) Vaakkriti: Sanskrit tokenizer. In: Proceedings of the 3rd international joint conference on natural language processing
Polykretis C, Kalogeropoulos K, Andreopoulos P, Faka A, Tsatsaris A, Chalkias C (2019) Comparison of statistical analysis models for susceptibility assessment of earthquake-triggered landslides: a case study from 2015 earthquake in Lefkada Island. Geosciences 9(8):350
Qian L, Luo Z, Du Y, Guo L (2009) Cloud computing: an overview. In: IEEE international conference on cloud computing. Springer, Berlin, pp 626–631
Rathod SG, Sondur S (2012) English to Sanskrit translator and synthesizer (ETSTS). Int J Emerg Technol Adv Eng 2(12):379–383
Rimal BP, Choi E, Lumb I (2009) A taxonomy and survey of cloud computing systems. In: INC, IMS and IDC, pp 44–51
Sachin K (2007) Sandhi splitter and analyzer for Sanskrit (with reference to AC sandhi). Mphil Thesis (SCSS), JNU
Schwartze C, Kralisch S, Flügel WA (2011) Geospatial virtual appliances using open source software. In: International symposium on environmental software systems. Springer, Berlin, pp 154–160
Sennrich R, Haddow B (2016) Linguistic input features improve neural machine translation. Preprint arXiv:1606.02892
Shi W, Lu Y, Li Z, Engelsma J (2011) Sharc: a scalable 3D graphics virtual appliance delivery framework in cloud. J Netw Comput Appl 34(4):1078–1087
Shobana M, Sabitha R, Karthik S (2020) An enhanced soft computing-based formulation for secure data aggregation and efficient data processing in large-scale wireless sensor network. Soft Comput 2020:1–12
Siddique N, Adeli H (2013) Computational intelligence: synergies of fuzzy logic, neural networks and evolutionary computing. Wiley, Berlin
Singh M, Kumar R, Chana I (2019a) Neural-based machine translation system outperforming statistical phrase-based machine translation for low-resource languages. In: 2019 12th international conference on contemporary computing (IC3). IEEE, New York, pp 1–7
Singh M, Kumar R, Chana I (2019b) Neuro-FGA based machine translation system for Sanskrit to Hindi language. In: 2019 international conference on innovative sustainable computational technologies (CISCT). IEEE, New York, pp 1–6
Skadiņš R, Tiedemann J, et al. (2012) Letsmt!: a cloud-based platform for do-it-yourself machine translation. In: Proceedings of the ACL 2012 system demonstrations. Association for Computational Linguistics, pp 43–48
Sun K, Mou S, Qiu J, Wang T, Gao H (2018) Adaptive fuzzy control for nontriangular structural stochastic switched nonlinear systems with full state constraints. IEEE Trans Fuzzy Syst 27(8):1587–1601
Upadhyay Pankaj KA, Chandra Jaiswal Umesh (2014) Transish: translator from Sanskrit to English-A rule based machine translation. Int J Comput Appl 4(5):2277–4106

Vasiljevs A, Skadiņš R, Tiedemann J (2011) Letsmt!: cloud-based platform for building user tailored machine translation engines. In: Proceedings of the 13th machine translation summit, pp 507–511

Vaswani A, Bengio S, Brevdo E, Chollet F, Gomez AN, Gouws S, Jones L, Kaiser Ł, Kalchbrenner N, Parmar N, et al. (2018) Tensor2tensor for neural machine translation. Preprint arXiv:1803.07416

Venugopal A, Zollmann A (2009) Grammar based statistical MT on hadoop: an end-to-end toolkit for large scale pscfg based mt. Prague Bull Math Linguist 91:67–78

Xing Y, Zhan Y (2012) Virtualization and cloud computing. In: Zhang Y (ed) Future wireless networks and information systems. Springer, Berlin, Heidelberg, pp 305–312

Yager RR, Zadeh LA, Kosko B, Grossberg S (1994) Fuzzy sets, neural networks and soft computing. Technical report

Zadeh LA (1996) Fuzzy logic, neural networks, and soft computing. Communication of the ACM, vol 37, no 3, pp 77–83

Zhang Z, Hong WC (2019) Electric load forecasting by complete ensemble empirical mode decomposition adaptive noise and support vector regression with quantum-based dragonfly algorithm. Nonlinear Dyn 98(2):1107–1136