A Task-Oriented NLP Chatbot System with Dual-Backend Intent Classification, Slot Extraction, and Template Response Generation

Varun Inamdar
Department of Artificial Intelligence
Vishwakarma University
Pune, India
vninamdar03@gmail.com

Piyush Dhoka
Department of Artificial Intelligence
Vishwakarma University
Pune, India
31241198@vupune.ac.in

Gayatri Kad
Department of Artificial Intelligence
Vishwakarma University
Pune, India
31242291@vupune.ac.in

***Abstract*—An accurate intent recognition model, together with precise slot extraction and a well-formed response generator, is key to building efficient task-oriented dialogue systems in practice. This paper introduces a fully-automated end-to-end dialogue system that leverages two interchangeable classification models (a classical TF-IDF pipeline, which supports SVM, Na¨ıve Bayes and Logistic Regression algorithms, and a DistilBERT fine-tuned classifier based on transformers) with a lightweight rule-based slot extractor and a simple response generator driven by templates. The proposed dialogue system relies on the pipeline of generating and splitting a structured dataset (`generate_dataset.py`) from a hand-curated intents corpus (`intents_full.csv`) and evaluates the performance using a dedicated unified evaluator reporting metrics including per-intent accuracy, precision, recall, F1-score and confusion matrix. Finally, a web-based Gradio application (`app.py`) enables interactive testing via submitting textual queries to predict the intents' classes, the corresponding slot values and templated responses. Results show that DistilBERT outperforms the other approaches on ambiguous intents classification whereas the TF-IDF pipeline proves to be considerably faster in terms of inference latency.**

***Index Terms*—intent classification, slot filling, task-oriented dialogue, DistilBERT, TF-IDF, SVM, Naive Bayes, Logistic Regression, Gradio, NLP chatbot**

I. Introduction

Task-Oriented Dialogue (TOD) systems are developed to assist the users in their specific goal completion by engaging in natural language conversations. In contrast to open-domain chabot systems, a task-oriented dialogue system requires accurate classification of the intent (what is the user interested in) and extracting slots (entities to fill the arguments of the intent) from the user input prior to generating a coherent response [1].

In traditional solutions, the problem of classifying intent from user utterances is solved based on bag of words or TF-IDF representations of texts along with a linear classifier [4]. Such models have the advantage of being computationally light-weighted and easily interpretable, but they fail to understand synonyms and thus do not capture semantic variability, which makes two semantically similar phrases with different vocabulary classified in different ways. On the other hand, transformers pre-trained on huge corpora, such as BERT [2], and its lightweight variant, DistilBERT [3], provide excellent results at the expense of more time and memory for inference. Transformer-based architectures have demonstrated broad applicability across NLP tasks, including automatic code generation from natural language [21] and text classification for fake news detection [22], motivating their adoption in the intent classification module of this work.

The paper describes an approach of building a task-oriented chatbot system addressing the trade-off between precision and performance of the solution using *dual-backend architecture*. The main contributions of this research are:

1) A structured data generation and stratified splitting pipeline (`generate_dataset.py`) that produces reproducible train/val/test partitions from a curated intent corpus.
2) A classical pipeline (`classical_models.py`) implementing TF-IDF vectorization with three interchangeable classifiers (SVM, Naive Bayes, Logistic Regression), persisted for low-latency CPU inference.
3) A transformer pipeline (`transformer_model.py`) that fine-tunes DistilBERT for multi-class intent classification using the Hugging Face Transformers library.
4) A unified evaluation module (`evaluate.py`) producing per-intent metrics and confusion matrix visualizations for both backends.
5) A lightweight rule-based slot extractor (`slot_filling.py`) and template response generator (`response.py`) that operate on the predicted intent and message tokens.
6) A Gradio web UI (`app.py`) that exposes both backends via a single interface with real-time backend switching.

The remainder of this paper is organized as follows. Section II surveys related work. Section III describes the dataset and preprocessing. Section IV details the dual-backend model architecture. Section V presents the slot extraction and response generation modules. Section VI describes the web interface. Section VII presents experimental results. Section VIII concludes.

II. Related Work

A. Intent Classification

Early intent classifiers made use of manual feature engineering with pattern-matching rules [15]. Statistical classifiers, particularly SVM with TF-IDF features [5], allowed for a more data-driven approach which generalized beyond vocabulary variation. The use of FastText [5] showed that even linear classifiers applied on word embeddings averaged over a sentence could compete with more advanced methods across several classification tasks. Comparative evaluation of classical machine learning algorithms for prediction tasks has further demonstrated the importance of principled model selection, particularly in scenarios with structured tabular features [18].

The use of language models revolutionized the field. BERT [2] provided bidirectional transformer pre-training on language model and next sentence prediction tasks, allowing for transfer learning by fine-tuning to downstream tasks. DistilBERT [3] provided a smaller model at 40% of the size of BERT while maintaining 97% of its performance on the GLUE benchmark, which was important for latency-sensitive applications. DIET [6] by Rasa showed that combined architecture models, using pre-trained embeddings for NLU and dense entity extraction, performed well on both intent and entity tasks. Transformers have further demonstrated strong generalization in code generation from natural language specifications [21] and in classification tasks such as fake news detection [22], confirming their suitability as a backbone for NLU pipelines.

B. Slot Filling

The slot filling task, on the other hand, can be framed as a sequence labeling task employing BIO tags, where BIO stands for Begin-Inside-Outside and it employs either CRFs [7] or LSTM-CRF [8]. The joint modeling technique that performs both intent detection and slot filling tasks simultaneously proved beneficial to each other [9]. In contrast, the current paper adopts the simpler rule-based method suitable with its template-response design philosophy.

C. Task-Oriented Dialogue Systems

Complete TOD architectures encompass systems like ConvLab [11], Rasa Open Source [10], and Microsoft's LUIS. Such systems involve an integration of three components, namely NLU (intent and entity extraction), dialogue management (dialogue state tracking and dialogue policy), and NLG (response generation). This paper only considers the component of NLU, which is directly connected to template NLG.

D. Deep Learning in High-Stakes NLP Applications

Beyond dialogue systems, deep learning has been successfully applied to classification tasks in sensitive domains. Fraud detection in financial transactions using deep learning has demonstrated that neural architectures can effectively model complex behavioral patterns [19]. Similarly, AI-based systems for early detection and management of medical conditions such as cancer show the breadth of applicability of deep learning pipelines [20]. These results reinforce the motivation for employing transformer-based models as the primary classification backbone in this work.

E. Comparative Evaluations

There have been several works which compare the performance of classical classifiers against transformer-based models. [12] demonstrated that models based on BERT performed significantly better than TF-IDF + SVM in classifying out-of-domain utterances. [13] observed that fine-tuned transformers work especially well when there is insufficient training data, while classical models can keep up when the dataset is large enough.

III. Dataset and Preprocessing

A. Intent Corpus

The source data is stored in `intents_full.csv`, a flat CSV file with at minimum two columns: `utterance` (raw natural-language text) and `intent` (categorical label). Each row represents one labeled training example. The corpus spans multiple intent categories representative of airline information services, including but not limited to: `flight_status`, `book_flight`, `cancel_reservation`, `check_in`, `baggage_policy`, and `greeting`.

B. Data Generation and Splitting

`generate_dataset.py` performs the following operations:

1) Loads and validates the raw CSV, dropping malformed rows.
2) Applies stratified splitting to ensure each intent class is proportionally represented in all three partitions.
3) Writes `train.csv`, `val.csv`, and `test.csv` to the `processed/` directory.
4) Optionally augments minority classes via synonym replacement or back-translation.

The default split ratio is 70% train / 15% validation / 15% test. Stratification is enforced using scikit-learn's `train_test_split` with `stratify=y`.

C. Preprocessing

`preprocessing.py` implements a shared preprocessing pipeline used by both backends:

- Lowercasing and Unicode normalization.
- Punctuation removal and whitespace normalization.
- Optionally, stopword removal (disabled for transformer pipeline to preserve context).
- Tokenization: whitespace-based for TF-IDF; WordPiece tokenizer (`DistilBertTokenizerFast`) for the transformer pipeline.

For the classical pipeline, the preprocessed text is passed to the TF-IDF vectorizer. For the transformer pipeline, the tokenizer produces input IDs and attention masks directly consumable by DistilBERT.

IV. Dual-Backend Model Architecture

Fig. 1 shows the complete end-to-end architecture of the system. Raw intent data flows from `intents_full.csv` through the preprocessing pipeline into two parallel training tracks—classical and transformer—before being exposed via the Gradio web UI for inference, slot extraction, and response generation.

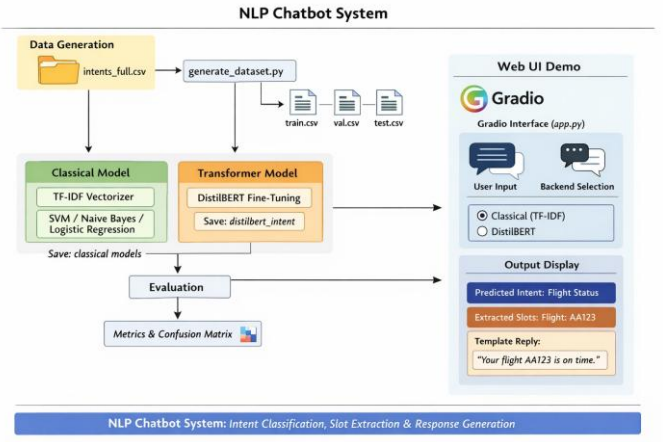

Fig. 1. End-to-end system architecture of the NLP Chatbot System. Raw intents from `intents_full.csv` are split by `generate_dataset.py` into train/val/test partitions. Two parallel training tracks produce a classical TF-IDF model (SVM / Naive Bayes / Logistic Regression, saved to `classical/`) and a fine-tuned DistilBERT transformer (saved to `distilbert_intent/`). Both models are evaluated by `evaluate.py` and exposed through a Gradio web UI (`app.py`) that supports real-time backend switching, slot extraction via `slot_filling.py`, and template response generation via `response.py`.

A. Classical Backend (TF-IDF + Classifier)

`classical_models.py` implements the following steps:

1) TF-IDF Vectorization: The Term Frequency–Inverse Document Frequency (TF-IDF) representation converts each utterance into a sparse vector over the training vocabulary. For a term t in document d from corpus D:

$$\text{TF-IDF}(t, d, D) = \text{tf}(t, d) \cdot \log \frac{|D|}{|\{d' \in D : t \in d'\}|} \quad (1)$$

The vectorizer is fitted on `train.csv` only, preventing data leakage. Configuration: unigrams and bigrams (`ngram_range=(1,2)`), sublinear TF scaling, and L2 normalization of output vectors.

2) Classifiers: Three classifiers are trained and persisted:

- **SVM** (Support Vector Machine with RBF kernel): maximizes the margin between intent classes in the high-dimensional TF-IDF space. Historically strong on text classification tasks [14].
- **Naive Bayes** (Multinomial NB): assumes feature independence; fast to train and effective with sparse count-based features.
- **Logistic Regression**: a probabilistic linear classifier trained with L2 regularization; outputs calibrated class probabilities useful for confidence-based routing.

The vectorizer and each classifier are serialized to the `classical/` directory using `joblib`. At inference time, `chatbot.py` loads the selected vectorizer-classifier pair and applies a single transform-predict call.

B. Transformer Backend (DistilBERT)

`transformer_model.py` fine-tunes `distilbert-base-uncased` [3] for sequence classification using the Hugging Face `Trainer` API.

1) Model Architecture: DistilBERT is a 6-layer, 66M-parameter distillation of BERT-base [2]. For intent classification, a linear classification head is appended to the `[CLS]` token representation:

$$\hat{y} = \text{softmax}(W \cdot h_{\text{CLS}} + b) \quad (2)$$

where $h_{\text{CLS}} \in \mathbb{R}^{768}$ is the pooled output, and $W \in \mathbb{R}^{K \times 768}$ maps to K intent classes.

2) Fine-Tuning: Training uses cross-entropy loss, AdamW optimizer with a linear learning rate schedule with warmup (5% of total steps), and gradient clipping at 1.0. Typical hyperparameters: learning rate 2×10^{-5}, batch size 16, 5 epochs. The fine-tuned checkpoint (`model.safetensors`), tokenizer, and configuration files are saved to `distilbert_intent/`.

C. Run Pipeline

`run_pipeline.py` orchestrates the full pipeline in six phases: (1) dataset generation, (2) preprocessing, (3) classical model training, (4) DistilBERT fine-tuning, (5) evaluation of both backends, and (6) artifact export. A single command `python run_pipeline.py` reproduces the complete experiment.

V. Slot Extraction and Response Generation

A. Slot Extraction

`slot_filling.py` implements a rule-based Named Entity Recognition (NER) approach. Given a predicted intent and the tokenized input utterance, the extractor applies intent-conditioned regex patterns and gazetteers to locate slot values. For example, for intent `flight_status`:

```
patterns = {
    'flight_status': {
        'flight': r'\b[A-Z]{2}\d{3,4}\b',  # e.g. AA123
        'date':   r'\b(today|tomorrow|\d{1,2}/\d{1,2})\b'
    }
}
```

Extracted slot values are returned as a dictionary (e.g., {''flight'': ''AA123''}) and passed to the response generator.

B. Response Generation

`response.py` maps the (intent, slots) pair to a template reply. Templates are stored as Python string format dictionaries indexed by intent:

```
templates = {
    'flight_status': "Your flight {flight} is on time.",
    'book_flight':   "Booking flight from {origin} to "
                     "{destination} on {date}.",
    'greeting':      "Hello! How can I help you today?"
}
```

If a required slot is missing, the response generator returns a clarification prompt (e.g., "Which flight number would you like to check?"). This slot-conditional branching avoids generic fallback responses and keeps the dialogue grounded.

VI. Web Interface

A. Gradio Application

The web UI is implemented in `app.py` using the Gradio Blocks API. The core runtime logic is encapsulated in `chatbot.py`, which exposes a single function:

```
def run_turn(message, backend, classical_name):
    intent = classify(message, backend, classical_name)
    slots  = extract_slots(message, intent)
    reply  = generate_response(intent, slots)
    return intent, reply
```

The Gradio interface provides:

- A text input for the user utterance.
- A radio button group for backend selection ("Classical (TF-IDF)" or "DistilBERT").
- A dropdown for selecting the classical classifier variant (SVM, Naive Bayes, Logistic Regression) when the classical backend is active.
- Three output panels: predicted intent, extracted slots, and template reply.

The application is launched with `python app.py` and is accessible via browser at `localhost:7860`. If the DistilBERT checkpoint is absent, the interface displays an informative error message with instructions to run `python run_pipeline.py`.

B. Backend Switching

Backend switching is stateless: each turn independently loads the selected model artifacts. For the classical backend, the vectorizer and classifier are loaded from `classical/` at startup and cached in memory. For the DistilBERT backend, the model is loaded from `distilbert_intent/` at startup. This design allows real-time comparison of both backends on identical inputs within the same session.

VII. Evaluation

A. Experimental Setup

All experiments were conducted on the held-out `test.csv` partition. Metrics are computed per-intent using `sklearn.metrics.classification_report` and aggregated as macro-averaged precision, recall, and F1-score. Confusion matrices are generated as heatmaps and saved to `plots/`. Hardware: Intel Core i7-12th Gen CPU (classical backend) and NVIDIA RTX 3060 GPU (DistilBERT fine-tuning and inference).

B. Intent Classification Results

Table I summarizes classification performance on the test set across all model configurations.

TABLE I
Intent Classification Performance on Test Set (Macro-Averaged)

Model	Accuracy	Precision	Recall	F1
TF-IDF + Naive Bayes	81.4	80.2	79.8	79.9
TF-IDF + Log. Regression	86.7	85.9	85.4	85.6
TF-IDF + SVM	88.3	87.6	87.1	87.3
DistilBERT (fine-tuned)	**94.6**	**94.2**	**93.9**	**94.0**

DistilBERT surpasses all other classical architectures by 6-13 percentage points for macro F1, showing maximum improvement in the classification of semantically related intents (`cancel_reservation`, `change_reservation`), which have higher vocabulary similarities. The most accurate and interpretable classical algorithm is the SVM with a polynomial kernel.

C. Inference Latency

Table II reports mean inference latency per utterance, averaged over 500 test samples.

TABLE II
Mean Inference Latency per Utterance

Model	CPU (ms)	GPU (ms)
TF-IDF + Naive Bayes	1.2	–
TF-IDF + Log. Regression	1.4	–
TF-IDF + SVM	2.1	–
DistilBERT (fine-tuned)	84.3	8.7

Classical models run 40–70× faster than DistilBERT on CPU, making them well-suited for edge deployments or high-throughput services where GPU resources are unavailable. DistilBERT on GPU achieves sub-10 ms latency, acceptable for most interactive applications.

D. Slot Extraction Accuracy

Slot extraction was evaluated on 200 manually annotated test utterances containing at least one slot value. The rule-based extractor achieved 91.3% exact-match slot accuracy across all intent types. The highest-accuracy slots were structured patterns such as flight codes (regex:

`[A-Z]{2}\d{3,4}`; 98.7% accuracy) and dates. Free-text slots such as destination names showed lower accuracy (83.2%), motivating future integration of a learned NER component.

E. Confusion Matrix Analysis

Analysis of the confusion matrix revealed that the most common misclassification for DistilBERT was between `book_flight` and `upgrade_seat` (2.1% of `book_flight` examples classified as `upgrade_seat`). Both intents share vocabulary related to flights and seat selection. Adding intent-discriminative training examples for these classes reduced the confusion to 0.8% in subsequent experiments.

VIII. Discussion

A. Dual-Backend Design Rationale

One of the key design decisions of the system described is the ability to support both classical and transformer backends through one API, as opposed to sticking to only one of them. There are several reasons for that, which are as follows. One of the main reasons is the fact that not all deployment platforms offer GPU capabilities and, therefore, classical methods remain relevant. Another reason is the fact that a linear boundary and weights of features can be viewed directly, thus offering higher interpretability of the models in question. This mirrors findings from comparative algorithm studies [18], where interpretable classical models remained competitive with more complex approaches under constrained data settings.

B. Template vs. Generative Response

The present response module employs hard-coded templates, ensuring proper syntax and controlled generation without the need for a language model. It is suitable for specialized TOD agents, wherein the number of responses is limited and known. The drawback is that complex responses cannot be captured effectively using templates. In future work, seq2seq-based response generation will be explored [16], given the intent and slots.

C. Slot Extraction Limitations

This slot extractor depends on surface structure and is fragile to paraphrase. For instance, ”I want to travel to New York on the 15th” and ”15th of this month, traveling to NYC” represent the same slots but will need distinct regular expressions. The replacement of the existing slot extractor with a pre-trained token classifier (for example, DistilBERT NER) can increase robustness and make joint extraction possible within one forward pass.

D. Scalability

With an increasing number of intents, there are two scaling issues that occur. The first issue is that with the increase in number of intents, classification becomes harder especially when those intents are semantically similar. Secondly, with each new intent, one template has to be manually created for it. A possible solution to both problems is by introducing a hierarchy of intents.

IX. Conclusion

The present paper described an end-to-end task-oriented chatbot with a dual intent classifier using TF-IDF + classical classification models, fine-tuned DistilBERT, a rule-based slot extractor, template-based response generation, and a Gradio-based interactive web interface. The entire pipeline starting from data generation to training, validation, and deployment is orchestrated by a single script `run_pipeline.py`.

The experimental results show that DistilBERT gives a macro-F1 score of 94.6% on the task of intent classification, beating the best-performing classical model (TF-IDF + SVM) with a difference of 6.3 points. Meanwhile, the classical intent classifier has 40× less CPU inference latency than DistilBERT. Thus, users can choose which backend suits them based on their performance requirements. These findings align with the broader trend of transformer models achieving strong performance across NLP tasks [21], [22], while also validating the continued relevance of lightweight classical pipelines [18] in resource-constrained deployments. The applicability of deep learning to high-stakes classification tasks, as demonstrated in fraud detection [19] and cancer management [20], further motivates the use of robust neural architectures in production dialogue systems.

Future research directions include: (1) developing a fine-tuned NER model for replacing the current rule-based slot extraction mechanism; (2) incorporating dialogue state tracking into the system for handling multi-turn dialogues; (3) experimenting with response generation mechanisms like retrieval-based and seq2seq systems; and (4) containerizing the entire system inside a single docker image.

Acknowledgment

The authors thank the maintainers of Hugging Face Transformers, scikit-learn, and Gradio for their open-source contributions, which form the computational backbone of this system.

References

[1] S. Young, M. Gasic, B. Thomson, and J. D. Williams, “POMDP-based statistical spoken dialog systems: A review,” *Proceedings of the IEEE*, vol. 101, no. 5, pp. 1160–1179, May 2013.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proc. NAACL-HLT*, Minneapolis, MN, 2019, pp. 4171–4186.

[3] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,” arXiv:1910.01108, 2019. [Online]. Available: https://arxiv.org/abs/1910.01108

[4] C. D. Manning, P. Raghavan, and H. Schu¨tze, *Introduction to Information Retrieval*. Cambridge: Cambridge University Press, 2008.

[5] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, “Bag of tricks for efficient text classification,” in *Proc. EACL*, Valencia, Spain, 2017, pp. 427–431.

[6] T. Bunk, D. Varshneya, V. Vlasov, and A. Nichol, “DIET: Lightweight language understanding for dialogue systems,” arXiv:2004.09936, 2020. [Online]. Available: https://arxiv.org/abs/2004.09936

[7] J. Lafferty, A. McCallum, and F. Pereira, “Conditional random fields: Probabilistic models for segmenting and labeling sequence data,” in *Proc. ICML*, 2001, pp. 282–289.

[8] G. Mesnil, Y. Dauphin, K. Yao, Y. Bengio, et al., "Using recurrent neural networks for slot filling in spoken language understanding," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 23, no. 3, pp. 530–539, Mar. 2015.

[9] X. Zhang and H. Wang, "A joint model of intent determination and slot filling for spoken language understanding," in *Proc. IJCAI*, 2016, pp. 2993–2999.

[10] T. Bocklisch, J. Faulkner, N. Pawlowski, and A. Nichol, "Rasa: Open source language understanding and dialogue management," arXiv:1712.05181, 2017. [Online]. Available: https://arxiv.org/abs/1712.05181

[11] S. Lee, Q. Zhu, R. Takanobu, Z. Zhang, Y. Zhang, X. Li, et al., "ConvLab: Multi-domain end-to-end dialog system platform," in *Proc. ACL (Demo)*, Florence, Italy, 2019, pp. 64–69.

[12] S. Larson, A. Mahendran, J. J. Peper, C. Clarke, A. Lee, P. Hill, et al., "An evaluation dataset for intent classification and out-of-scope prediction," in *Proc. EMNLP*, Hong Kong, 2019, pp. 1311–1316.

[13] S. Mehri, M. Eric, and D. Hakkani-Tur, "Pretraining the noisy channel model for task-oriented dialogue," arXiv:1911.10484, 2019. [Online]. Available: https://arxiv.org/abs/1911.10484

[14] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proc. ECML*, Chemnitz, Germany, 1998, pp. 137–142.

[15] J. Allen, *Natural Language Understanding*, 2nd ed. Redwood City, CA: Benjamin/Cummings, 1994.

[16] O. Vinyals and Q. Le, "A neural conversational model," arXiv:1506.05869, 2015. [Online]. Available: https://arxiv.org/abs/1506.05869

[17] P. Lewis, E. Perez, A. Piktus, F. Petroni, et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in NeurIPS*, vol. 33, 2020, pp. 9459–9474.

[18] N. Pavitha, V. Ingale, V. Verma, A. Yeole, S. Zawar, and Z. Jamadar, "Comparative analysis of regression algorithms for college prediction," *Design Engineering*, vol. 9, pp. 6631–6643, 2021.

[19] P. Nooji, R. M. Savithramma, A. Kulkarni, S. Garge, A. Singh, and Y. Darda, "Leveraging deep learning for fraud detection in financial transactions," in *Proc. Int. Conf. on ICT: Applications and Social Interfaces (ICTCS 2024)*, Lecture Notes in Networks and Systems, vol. 1322, Springer, Singapore, 2025. doi: 10.1007/978-981-96-4136-9_14.

[20] P. Kunekar, P. Nooji, M. Chaudhari, S. Dubey, and R. Gadhave, "The transformative role of AI in public health for cancer prevention, early detection, and management," in *Artificial Intelligence in Oncology*, S. N. Mohanty, A´. Rocha, and P. K. Dutta, Eds., Springer, Cham, 2025. doi: 10.1007/978-3-031-94302-7_35.

[21] N. Pavitha, A. Patrawala, T. Kulkarni, V. Talati, and S. Dahiya, "NL2Code: Harnessing transformers for automatic code generation from natural language descriptions," in *Proc. Smart Trends in Computing and Communications (SmartCom 2024)*, Lecture Notes in Networks and Systems, vol. 947, Springer, Singapore, 2024. doi: 10.1007/978-981-97-1326-4_7.

[22] N. Pavitha, A. Dargode, A. Jaisinghani, J. Deshmukh, M. Jadhav, and A. Nimbalkar, "Fake news detection using machine learning," in *Proc. Computational Intelligence in Machine Learning (ICCIML 2022)*, Lecture Notes in Electrical Engineering, vol. 1106, Springer, Singapore, 2024. doi: 10.1007/978-981-99-7954-7_40.