Enhancing Customer Service Chatbots with Context-Aware NLU, Selective Attention and Multi-task Learning for Improved Intent Classification

Anonymous ACL submission

Abstract

Customer service chatbots serve as conversational systems aimed at addressing customer queries. By directing customers to automated workflows, these chatbots enable faster query resolution. A crucial aspect of this process is classifying the customer's intent. Most existing intent classification models in the customer care domain rely solely on customer queries for prediction, which can be ambiguous and result in reduced model accuracy. For example, a query like "I did not receive my package" could indicate a delayed order, or a delivered order that the user failed to receive, each requiring a different resolution approach. Utilizing additional information, such as the customer's order delivery status, can enhance intent prediction accuracy. In this study, we introduce a context-aware NLU architecture that incorporates both the customer query and the customer's past order history as context. A novel selective attention module extracts relevant context features, leading to improved model accuracy. We also propose a multi-task learning paradigm for the effective utilization of different label types, one based only on user query and the other based on full conversation with human agent. Our suggested method, Multi-Task Learning-Contextual NLU with Selective Attention Weighted Context (MTL-CNLU-SAWC), demonstrated a 4.8% increase in top 2 intent accuracy score compared to the baseline model that only uses user queries, and a 3.5% improvement over existing state-of-the-art models combining query and context.

1 Introduction

Conversational agents have become an essential part of modern life, playing crucial roles in various activities such as playing music, ordering food, booking flight tickets (Handoyo et al. 2018), and handling banking tasks (Kiran et al. 2023). These agents have significantly impacted nearly every aspect of our lives. In the realm of customer care, they have efficiently resolved user queries, leading to increased customer satisfaction and saving companies millions of dollars. One of the core components of these conversational agents is Natural Language Understanding (NLU). The primary task of NLU is to comprehend the semantic meaning of a user's utterance, which is commonly referred to as intent classification. Only after this task is achieved can the conversational agent assist the user with subsequent steps. Our work concentrates on intent classification within the customer care domain.

In recent years, Transformers (Vaswani et al. 2017; Wolf et al. 2020) have gained popularity in model pre-training (Howard and Ruder 2018) and have achieved state-of-the-art results in numerous NLU tasks. They have been widely adopted for the intent classification task across various domains and have generally exhibited strong performance. State-of-the-art solutions, such as those by Rafiepour and Sartakhti 2023, Chen et al. 2019, and Lorenzo et al. 2021, employ transformer-based architectures for intent classification. Specifically, in the customer care domain, Wang et al. 2021 and Senese et al. 2020 implement intent classification using transformer models. A common theme among these works is that they only consider the user's utterance for intent classification. This approach is effective when the user provides a detailed description of the issue. However, in the customer care domain, user utterances often lack sufficient information and can be vague. Table 1 lists some examples of vague user utterances where context can be helpful. In such cases, it is crucial for the model to leverage additional contextual cues for successful intent prediction.

One study that utilizes context to classify user intents is (Gupta et al. 2019). In this work, the model is trained using multiple signals, such as previous intents, slots and utterances over a variable context window, in addition to the current user

utterance. Here, context refers to the additional explicit input provided by the user. Another study, (Lu et al. 2019), leverages user profile information, like membership status, as context to generate chatbot responses. However, neither of these works incorporates information from the user's transaction data as context. In our work, we use the user's previous transactions to calculate features, such as order delivery status and order cancellation status, and combine them with the user query to predict intents.

Our context features can be either categorical or numerical, while the user query is textual. Therefore, it is essential to develop a mechanism to combine data from these different modalities. One notable work on this topic, (Kaas et al. 2020), proposes a method of combining neural BERT representations with hand-crafted features via stacked generalization. Ostendorff et al. 2019 also combines data from different modalities by concatenating BERT embeddings of a book description, metadata about the book (categorical and numerical), and knowledge graph embeddings of the author to classify books into genres. In another significant work, Gu and Budhkar 2021 conducts a comprehensive study of various techniques used to combine features from different modalities. Their performances are compared across different downstream tasks using various datasets. Promising architectures, which have performed the best in at least one of the tasks, include Unimodal (tokenizes numerical and categorical features and feeds them to the transformer model along with text), Concat (concatenates text embedding with raw tabular data), MLP+Concat (passes tabular features through an MLP block before concatenating with text embedding), Gating (Rahman et al. 2020), and weighted sum of feature vectors from different modalities. In our work, we have employed an attention mechanism to dynamically attend to each context feature based on the user query and context. The attention-weighted context vector is then combined with the query embedding to make intent predictions.

For deep learning models to perform optimally, high-quality labeled data is essential. However, in the customer care domain, as previously noted, we often encounter ambiguous user requests. Consequently, in many instances, labels derived solely from user utterances may not align with the actual labels. Some prior studies addressing the issue of learning from noisy labels in real-world scenarios include (Wei et al. 2021), (Frénay and Verleysen 2013), and (Huang et al. 2023). To tackle this challenge, we introduced conversation labels in addition to utterance labels. We developed an architecture utilizing the multi-task learning framework to effectively leverage both. The key contributions of this work are as follows:

Utterance	Possibilities
order cancelled	unclear whether user wants an order cancelled or is complaining about an order cancelled by store
my order	will require different resolution depending on whether the user is talking about a delayed order or missing order
not received	not clear whether the user is talking about order or refund

Table 1: Vague utterances where context can help

- Crafting features from the user's raw context to streamline the model's learning process
- Implementing an attention mechanism that enables the model to dynamically focus on context features based on the user query and context
- A labeling approach that considers users' latent and explicit intent
- An architecture employing the multi-task learning paradigm that effectively leverages the aforementioned labeling strategy, leading to overall performance enhancement.

2 Methodology

2.1 Combining order level, item level, and handcrafted contextual features

The context data obtained from a customer's transaction details includes information in the form of order-level features, such as order placement time, the number of items in the order, store number, and delivery fulfillment type, among others. Additionally, it comprises item-level features like delivery status for each item, item cancellations, and refund requests for individual items, and so on. We combined item-level features corresponding to an order to create new features, including "number of items delivered", "time difference between the last delivered item and user chat", "time difference between the last shipped item and user chat" and

more. These handcrafted features were further employed to generate additional features, such as "are any items left to be delivered", "are any items left to be shipped" and so on. For instance, the feature "are any items left to be delivered" is produced by verifying if the number of ordered items equals the number of delivered items. These handcrafted features were developed to facilitate the model's learning of complex feature interactions and have proven to enhance performance, as evidenced in Table 4. More examples of such handcrafted features can be found in Table 6 in Appendix F.

For all the architectures discussed in the subsequent sections, we have utilized Bidirectional Encoder Representations from Transformers (BERT)(Devlin et al. 2018) as the encoder to obtain embeddings from textual data.

2.2 Baseline Model

In the baseline model, the user utterance is input into a pre-trained BERT model, and the resulting embedding is passed through an MLP (Multi-Layer Perceptron) block, consisting of two hidden layers. The first linear layer is followed by a ReLU (Fukushima 1975) activation function layer, while the second linear layer is followed by a softmax (Bridle 1989) layer that outputs intent probabilities. The architecture's details can be found in Figure 1. Given dataset D = (x_i , y_i) with N different classes and M examples, we fine-tune BERT and train the MLP block layers. The output probability from the model can be represented as follows:

$$p(y|h_i) = \text{softmax}(h_i) \in R^N \tag{1}$$

where $h_i \in R^N$ is the output from the last layer of the MLP block before the application of softmax, for the i-th example x_i. The model parameters θ are trained on D with cross-entropy loss.

$$\theta^* = \underset{\theta}{\text{argmin}}\, \mathcal{L}_{ce}(D;\theta) \tag{2}$$

Cross entropy loss is defined as:

$$\mathcal{L}_{ce} = -\sum_{i=1}^{M}\sum_{c=1}^{N} y_{i,c}\log(p_{i,c}) \tag{3}$$

where $p_{i,c}$ is the predicted probability of the i-th example belonging to class c, and $y_{i,c} \in \{1, 0\}$, depending on whether c is the true class for the i-th example or not. $\hat{y}$ in Figure 1 denotes the intent class with the maximum probability.

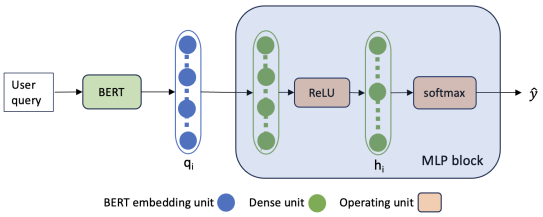

Figure 1: Baseline Model - User query passed through pre-trained BERT model and subsequent MLP block

2.3 Architecture to combine user utterance and context

The contextual features for the model are initially generated from the user's past transaction data. The procedure for creating these features is detailed in Section 2.1. These contextual features then undergo an additional preprocessing step, which includes min-max normalization and imputation of missing values. As with our baseline model, the user's query is input into a pre-trained BERT model. The BERT model's embedding is combined with the pre-processed contextual features, and the resulting combined embedding is fed through an MLP block. Among the numerous techniques for combining query and context, the most straightforward approach is to concatenate the query and context embeddings. However, it is challenging for MLP layers to attend to relevant information in the context vector for a given query embedding. Consequently, the Concat (Gu and Budhkar 2021) model's performance is unsatisfactory.

2.3.1 Concat with Attention weighted Context (CAWC)

We utilize an attention-based feature weight generation mechanism in which attention weights are computed for each of the context features, considering both the context and query embeddings. This approach enables the model to concentrate on relevant features, significantly mitigating the issues associated with the concatenation model. Let q_i and c_i denote the query embedding vector and context vector, respectively, for the i-th example. The attention module receives q_i and c_i and produces an attention vector a_i with the same length as c_i. The weighted context vector $\tilde{c}_i$ is then determined by performing element-wise multiplication of a_i and c_i. Finally, the weighted context vector is concatenated with the query embedding, and the combined embedding is input to an MLP block as before. The

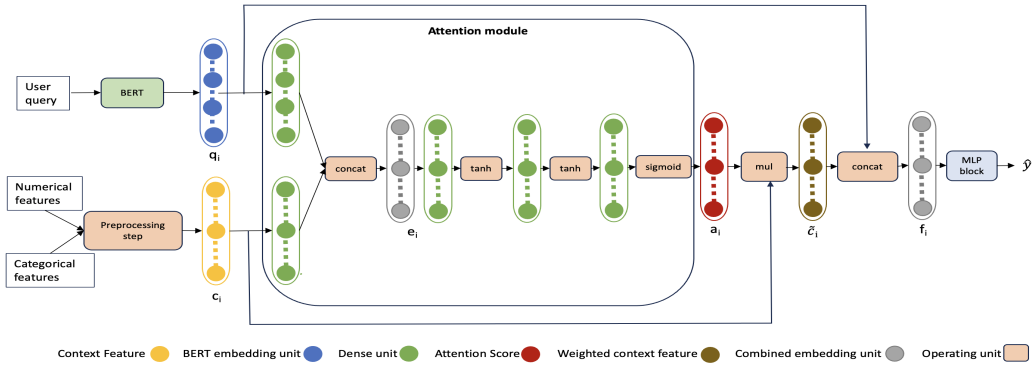

Figure 2: Concat with Attention weighted Context(CAWC) - Attention scores for each of the context features is calculated based on context and query vectors, using the attention module. Attention weighted context vector is then concatenated with query embedding

complete architecture can be seen in Figure 2.

$$a_i = \text{AttentionModule}(q_i, c_i) \tag{4}$$

$$\hat{c}_i = a_i \odot c_i \tag{5}$$

where $\odot$ represents element-wise multiplication

Attention Module: As previously mentioned, the attention module accepts the context vector and query embedding vector as inputs and generates attention scores for each context feature. Within the attention module, both the query and context vectors are passed through linear layers represented by W_q and W_c, respectively, and subsequently concatenated to form a combined vector denoted as e_i. The vector e_i is then fed through two linear layers, symbolized by W_{l1} and W_{l2}, with the number of neurons roughly halved in each hidden layer. The tanh activation function is applied after each linear layer. Following this, the resulting vector is passed through another linear layer, denoted by W_{l3}, which has an output vector length equal to that of the context vector. The sigmoid activation function, σ, is then applied to restrict each value between 0 and 1. The resulting attention vector is represented by a_i.

$$e_i = \text{concat}(W_q q_i, W_c c_i) \tag{6}$$

$$a_i = \sigma(W_{l3} \tanh(W_{l2} \tanh(W_{l1} e_i))) \tag{7}$$

where σ represents the sigmoid function

2.4 Labelling Strategy

We have discussed how incorporating contextual information can assist the model in predicting more accurate intents, especially when the user utterance is ambiguous. Similarly, having context information aids in the proper annotation of data, which subsequently enhances the model's performance. In our scenario, labeling examples solely on user utterances might result in incorrect labels when the user query is vague. Nevertheless, the accurate approach involves examining the user's query along with all its contextual features for labeling the examples. However, this method is extremely time-consuming and not scalable for large datasets, such as customer care. To overcome this, we use the entire user-agent (human) conversation as a substitute for the user's context information since this data is relatively easier to label. This conversation data is an appropriate proxy because most of the context information that can be derived from the user's transaction history is often mentioned during the user-agent conversation. Therefore, we have two types of labels for each example:

- **Utterance Label:** tagged based solely on the user's utterance and is intended to capture the user's explicit intent
- **Conversation Label:** tagged based on the entire user-agent conversation and captures the user's latent intent.

The subsequent step is to efficiently utilize the two types of labels for training our model. As previously mentioned, training models with utterance labels might be sub-optimal since these labels were tagged based solely on the user utterance. Likewise, training models with only conversation labels might not always be effective, as doing so could deviate from what the user has explicitly stated. For instance, when the user types "contact customer care," the utterance label would be "agent contact."

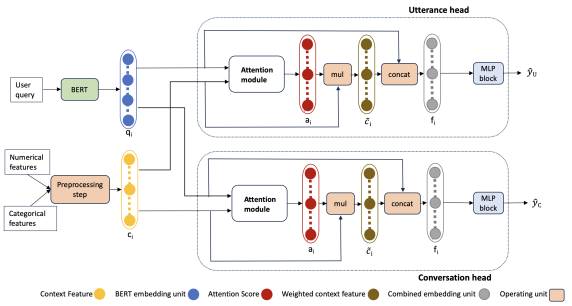

Figure 3: MTL-CNLU: Model with two heads trained jointly in a multi-task learning setting using both, utterance label and conversation label

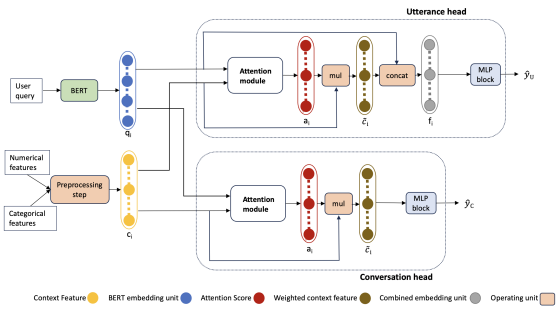

Figure 4: MTL-CNLU-AWC: Only attention weighted context vector used to predict conversation intent

However, the conversation label would generally indicate the user's latent intent, which could be about a "refund" and is only discerned by the human agent after further interactions with the user. In such cases, predicting "agent contact" leads to a better user experience. To address this dilemma, we decided to employ the multi-task learning approach to train our model.

2.5 Multi Task learning paradigm for Contextual NLU (MTL-CNLU)

Multi-Task Learning (MTL) is a training framework employed to exploit valuable information within multiple related tasks, thereby enhancing the generalization performance across all tasks (Zhang and Yang 2021; Zhang et al. 2022). In our context, we define one task as the classification of utterance intent and another task as the classification of conversation intent. The utterance label and conversation label serve as the ground truths for the first and second tasks, respectively. In the majority of multi-task learning models, the initial layers are shared among all tasks (Ruder 2017). In our scenario, the backbone BERT module is shared between the utterance head (accountable for predicting utterance intent) and the conversation head (accountable for predicting conversation intent). The parameters of the backbone BERT module, utterance head, and conversation head are denoted by ϕ_1, ϕ_{2U}, and ϕ_{2C}, respectively. The combined loss from the two heads is utilized to jointly update all three parameter sets and is represented by:

$$\mathcal{L}_{combined} = \mathcal{L}(D; \phi_1, \phi_{2U}, \phi_{2C}) \quad (8)$$

The combined loss is a weighted sum of the cross-entropy losses from the two heads.

$$\mathcal{L}_{combined} = \mathcal{L}_{ce}(Y_u, \hat{Y}_u) + \lambda \mathcal{L}_{ce}(Y_c, \hat{Y}_c) \quad (9)$$

where λ is a hyperparameter employed to balance the two losses. Y_u, Y_c, $\hat{Y}_u$, and $\hat{Y}_c$ represent the utterance labels, conversation labels, predicted utterance intents, and predicted conversation intents, respectively.

2.5.1 Top 2 intent selection

As discussed in Section 2.4, the correct intent to display to the user in response to a query such as "contact customer care" is "agent contact," as it precisely captures the user's explicit intent. Nonetheless, it is also crucial to reveal the user's implicit intent, which in this instance was about "refund," as evidenced by their context information. This helps to direct more customers to automated workflows. All architectures before MTL-CNLU had only one head, so to obtain the top 2 intents, we would choose the top 2 intents based on confidence scores directly from this head. In MTL-CNLU, the top 2 intents consist of the top intent from the utterance head and the top intent from the conversation head. Additionally, a metric is needed to assess model performance based on both predictions. For this purpose, models are evaluated on the top 2 score 3.2.2, as illustrated in Table 2.

2.6 Architectures for MTL-CNLU

2.6.1 MTL-CNLU

The pre-trained BERT module is the only component shared by both the utterance and conversation heads. Each head possesses its own query-context combining module, which maintains the same architecture as CAWC. The comprehensive architecture can be seen in Figure 3. The predicted intents from the utterance and conversation heads are represented by $\hat{y}_u$ and $\hat{y}_c$, respectively. All other notations remain unchanged from previous descriptions.

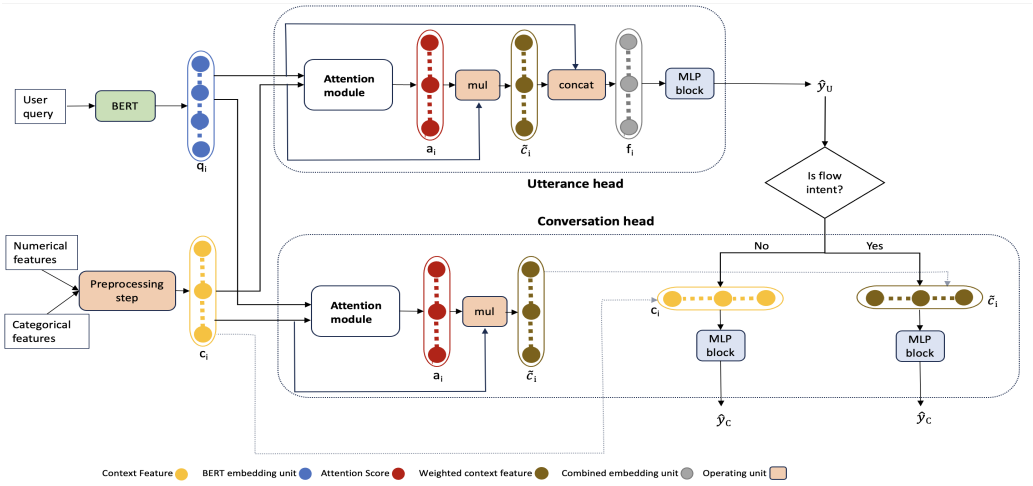

Figure 5: MTL-CNLU-SAWC: Attention weighted context vector used to predict conversation intents when user query corresponds to flow intent, otherwise context features used directly

2.6.2 MTL-CNLU with Attention Weighted Context (MTL-CNLU-AWC)

The concatenation of the query and weighted context vector for predicting intents works effectively for the utterance head. However, due to the context vector's sparse nature (attributed to the presence of categorical features) and its comparatively shorter length (the context vector has a length of 50, while the query vector has a length of 768), concatenating the query and context vectors causes the model to focus more on the query than the context. This results in suboptimal outcomes for the conversation head. For instance, our training data included an utterance "hello there" labeled with "greet" for utterance and "why order was cancelled" for conversation. Based on the context data also, "why order was cancelled" was deemed the appropriate intent since the order had been cancelled by the store due to items being out of stock. However, when trained on this data, our model formed a strong association between the utterance and conversation labels. Consequently, during inference, when a user with a latent intent of tracking their order status entered the query "hello there," the model predicted "greet" and "why order was cancelled" as the intents from the utterance and conversation heads, respectively, instead of "greet" and "where is my order".

Owing to findings like these and since the conversation head's primary objective is to predict the user's latent intent, the previous architecture was modified to remove the query-context combining module from the conversation head. Only the weighted context vector was fed into the MLP block to predict latent intents. The detailed architecture can be viewed in Figure 4.

Every user intent predicted by the model can be classified into one of two categories:

- **Flow intent:** Intents associated with a defined flow. When a user selects a flow intent, they can follow a series of predefined steps to resolve their query. Examples of flow intents include "where is my order", "why order was cancelled" and "where is my refund".
- **Non-flow intent:** Intents that do not have an associated flow. These include intents like "agent contact", "greet", "affirmative" and so on.

Utterance labels can be either flow or non-flow intents, while conversation labels are always flow intents.

2.6.3 MTL-CNLU with Selective Attention Weighted Context (MTL-CNLU-SAWC)

The attention weights, which are derived using both context and query embeddings as detailed in Section 2.3.1, assist the model in focusing on relevant context features. For instance, consider a user inputting the query "late." The context vector includes information that the user ordered two items: one was canceled by the store, and the other was delayed. The model must determine which of these two context features is more crucial: "are any items delayed" or "are any items canceled". This cannot be determined by the model based solely on context features. The query vector aids the model in focusing on the appropriate context feature. However, for utterances such as "contact customer care"

Architecture	Utterance Intent		Conversation Intent		Top 2 Score(%)
	Micro F1(%)	Macro F1(%)	Micro F1(%)	Macro F1(%)	
Text only (baseline)	78.65	75.80	-	-	86.12
Concat (Gu and Budhkar 2021)	80.14	77.28	-	-	87.34
MLP + Concat (Gu and Budhkar 2021)	80.28	77.66	-	-	87.23
Unimodal (Gu and Budhkar 2021)	79.66	76.01	-	-	86.14
Gating (Gu and Budhkar 2021; Rahman et al. 2020)	80.45	77.42	-	-	87.41
Weighted Sum (Gu and Budhkar 2021)	80.12	77.37	-	-	86.98
CAWC	81.5	78.71	-	-	88.38
MTL-CNLU	81.65	78.80	38.65	37.80	89.90
MTL-CNLU-AWC	81.54	78.81	41.78	38.95	90.44
MTL-CNLU-SAWC	**81.96**	**79.05**	**42.03**	**39.56**	**90.92**

Table 2: Results comparing performance of different models. The first half of the table contains results from the baseline model(text only) as well as the different SOTA models mentioned in (Gu and Budhkar 2021). The second half contains our models. The first 7 models have single headed architectures, therefore no comparison could be made with conversation label. Top 2 score is calculated based on Algorithm 2

and "talk to representative," the query embedding should ideally not influence the latent intent prediction, as there is no relevant information in the query regarding the user's latent intent. In fact, for all utterances where the explicit intent corresponds to a non-flow intent, the latent intent prediction should depend solely on the context vector. To accomplish this, we modify the architecture so that the context vector c_i is element-wise multiplied with the attention vector a_i only when the utterance head predicts a flow intent. In other cases, only c_i is fed as input to the conversation head. The logic for selectively applying attention is described in Algorithm 1 in Appendix A. Architecture details can be found in Figure 5. MTL-CNLU-SAWC is our final architecture and outperforms all others described thus far.

3 Experimental Setup

3.1 Dataset

The experimental data is sourced from the e-commerce customer care domain. Each example consists of a user utterance, context features, and two labels: utterance and conversation labels. Since the data used for experimentation is internal user data from an organization, it is not shared in this work. Context data is available for 70% of the examples in the form of the user's past transactions. For such examples, context features fed to the model are generated as described in Section 2.1. For examples without context features, a zero vector with a length equal to that of the context features is created and utilized as the context vector. Detailed statistics of our dataset can be found in Table 7 in Appendix G.

3.2 Evaluation metrics

3.2.1 Micro-F1 and Macro-F1 scores

The top 1 intent from the utterance head is compared to the utterance label, while the top 1 intent from the conversation head is compared to the conversation label. Micro-F1 and Macro-F1 scores are used as performance metrics for both. Micro-F1 is the same as accuracy, whereas Macro-F1 is equal to the average F1-score of the labels.

3.2.2 Top 2 score

As mentioned in Section 2.5.1, the top 2 score is a relevant metric for us. We consider the predicted intents from the utterance and conversation heads as the 1st and 2nd intents, respectively. In case both happen to be the same, the 2nd intent from the utterance head is considered as the 2nd intent. The logic for calculating the top 2 score can be found in Algorithm 2 in Appendix B.

4 Results

In Table 2, the evaluation metrics described in Section 3.2 are used to compare all of the proposed architectures in this paper, as well as those studied in (Gu and Budhkar 2021), on our test data. As demonstrated by the results, **MTL-CNLU-SAWC** is the best-performing model across all three metrics. Among the single-headed architectures, CAWC performs the best due to its effective integration of query and context information through its attention module. The MTL-CNLU-based architectures provide a performance boost in the top 2 accuracy scores, as the model becomes more adept at predicting the user's latent intent. This is also accompanied by a slight increase in the top 1 prediction, as the loss from the conversation

	Utterance	Context Information	utterance label	conversation label	Baseline	concat	MLP+concat/ Unimodal/ Gating/ Weighted Sum	CAWC	MTL-CNLU	MTL-CNLU-AWC	MTL-CNLU-SAWC
1	my order is late	items are overdue	order late	order late	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
2	cancel my order	items are yet to be delivered	cancel order	cancel order	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
3	order cancelled	store cancelled order	why order was cancelled	why order was cancelled	x	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
4	when will i receive rest of the items	some items were delivered while others were delayed	order late	order late	x	x	x	✓✓	✓✓	✓✓	✓✓
5	order help	items were missing from order	where is my order	missing items	✓	✓	✓	✓	✓✓	✓✓	✓✓
6	need to speak with agent	all items overdue	agent contact	order late	✓	✓	✓	✓	✓✓	✓✓	✓✓
7	hello	all items overdue	greet	order late	✓	✓	✓	✓	✓	✓	✓✓

Table 3: Qualitative analysis of different architectures. Double check mark (✓✓) indicates that model outputs cover both the labels, single check mark (✓) indicates that model outputs cover only one of the two labels and cross (x) indicates none of the labels are covered.

head helps the model learn better query embeddings. With MTL-CNLU-SAWC, the model can selectively use information from the user query to create a context vector that further enhances latent intent prediction.

Table 3 presents the results of a qualitative analysis of the different architectures by comparing their outputs for various examples. A double check mark (✓✓) indicates that the model's outputs cover both labels, a single check mark (✓) indicates that the model's outputs cover only one of the two labels, and a cross (x) indicates that none of the labels are covered. In example 1 and 2, the utterance is clear and the models are able to predict intents accurately based on just the utterance. For example 3, the baseline model, which is trained on just utterance, incorrectly predicts "cancel order" as the intent. Other models, with context information are able to correctly predict the intent. In example 4, the context contains information that some items are yet to be delivered and they are all overdue. The concat architecture as well as the architectures described in (Gu and Budhkar 2021) incorrectly predicts "missing items" for this. However, the attention based architecture is able to focus more on the information that the items were overdue and hence correctly predicts "order late" as the intent. For examples 5 to 7, utterance and conversation labels are different. For example 5, the single headed models are able to predict only the utterance intent while MTL-CNLU models are able to correctly predict both intents. For examples 6 and 7, where utterance intent corresponds to non flow intent, the query should ideally not influence the context vector at all. The MTL-CNLU model shows variable performance, getting conversation intent prediction correct for example 7 but wrong for example 8. This is because the attention module which depends on both query and context sometimes incorrectly focuses on the wrong context features. The MTL-CNLU-SAWC model gets the conversation intent correct for both of these cases.

5 Conclusion

In this paper, we have presented an effective approach for combining user queries and context information for the intent classification task in the customer care domain of e-commerce. Our proposed model outperforms both the baseline model, which only uses the user query, as well as other existing state-of-the-art models that aim to combine query and context. We experiment with different types of context features and create several features manually to simplify the learning process of the model. To address the problem of noisy labels, we also incorporate conversation labels in addition to utterance labels, and develop a method to effectively use both labels within the multi-task learning framework. Additionally, we show that selectively applying attention weights based on specific conditions can further improve model performance.

6 Limitations

In our current work, which focuses on the e-commerce customer care domain, we employ context information from the customer's transaction history in conjunction with the user query to predict customer intent. However, there is an untapped potential source of context information: the customer's website/app interaction data. This could be particularly valuable if the customer has recently visited help pages before engaging in a chatbot conversation.

Another limitation of our work is that it assumes the user utterance is the sole explicit input from the user to the chatbot. However, this assumption may not hold in the following scenario: the user enters a query, the chatbot generates intents as a response, the user finds the displayed intents unsatisfactory and chooses not to select any of them, and then the user types a new query that expands on the original one. Our current model disregards the previous query-response pair and considers the new query as the only explicit input to the model. As a result, the model may end up repeating its previous response, leading to a sub-optimal user experience. We plan to address both of these issues in our future work.

In our research, we have categorized each query (+context) into predefined intent classes. However, for the customer care domain, the class labels can be quite informative. Some examples of class labels are "where is my order", "where is my refund", "missing items" etc. Currently, we have not utilized the inherent information present in these label texts. As part of our future work, we also intend to explore different techniques to effectively leverage the information contained within the labels for our task.

References

John Bridle. 1989. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. *Advances in neural information processing systems*, 2.

Qian Chen, Zhu Zhuo, and Wen Wang. 2019. Bert for joint intent classification and slot filling. *arXiv preprint arXiv:1902.10909*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Benoît Frénay and Michel Verleysen. 2013. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems*, 25(5):845–869.

Kunihiko Fukushima. 1975. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3-4):121–136.

Ken Gu and Akshay Budhkar. 2021. A package for learning on tabular and text data with transformers. In *Proceedings of the Third Workshop on Multimodal Artificial Intelligence*, pages 69–73.

Arshit Gupta, Peng Zhang, Garima Lalwani, and Mona Diab. 2019. Context-aware self-attentive natural language understanding for task-oriented chatbots.

Eko Handoyo, M. Arfan, Yosua Alvin Adi Soetrisno, Maman Somantri, Aghus Sofwan, and Enda Wista Sinuraya. 2018. Ticketing chatbot service using serverless nlp technology. In *2018 5th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, pages 325–330.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

Zhizhong Huang, Junping Zhang, and Hongming Shan. 2023. Twin contrastive learning with noisy labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11661–11670.

Anders Kaas, Viktor Torp Thomsen, and Barbara Plank. 2020. Team disaster at semeval-2020 task 11: Combining bert and hand-crafted features for identifying propaganda techniques in news. In *SemEval 2020*. Association for Computational Linguistics.

Ajmeera Kiran, I. Jeya Kumar, P. Vijayakarthik, S.K Lokesh Naik, and T. Vinod. 2023. Intelligent chat bots: An ai based chat bot for better banking applications. In *2023 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–4.

Javier Lorenzo, Ignacio Parra, and MA Sotelo. 2021. Intformer: Predicting pedestrian intention with the aid of the transformer architecture. *arXiv preprint arXiv:2105.08647*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Yichao Lu, Manisha Srivastava, Jared Kramer, Heba Elfardy, Andrea Kahn, Song Wang, and Vikas Bhardwaj. 2019. Goal-oriented end-to-end conversational models with profile features in a real-world setting. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*, pages 48–55.

Malte Ostendorff, Peter Bourgonje, Maria Berger, Julian Moreno-Schneider, Georg Rehm, and Bela Gipp. 2019. Enriching bert with knowledge graph embeddings for document classification. *arXiv preprint arXiv:1909.08402*.

Mehrdad Rafiepour and Javad Salimi Sartakhti. 2023. Ctran: Cnn-transformer-based network for natural language understanding. *arXiv preprint arXiv:2303.10606*.

Wasifur Rahman, Md Kamrul Hasan, Sangwu Lee, Amir Zadeh, Chengfeng Mao, Louis-Philippe Morency, and Ehsan Hoque. 2020. Integrating multimodal information in large pretrained transformers. In *Proceedings of the conference. Association for Computational Linguistics. Meeting*, volume 2020, page 2359. NIH Public Access.

Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*.

Matteo Antonio Senese, Giuseppe Rizzo, Mauro Dragoni, and Maurizio Morisio. 2020. Mtsi-bert: a session-aware knowledge-based conversational agent. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 717–725.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962v2*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Peiyao Wang, Joyce Fang, and Julia Reinspach. 2021. Cs-bert: a pretrained model for customer service dialogues. In *Proceedings of the 3rd Workshop on Natural Language Processing for Conversational AI*, pages 130–142.

Jiaheng Wei, Zhaowei Zhu, Hao Cheng, Tongliang Liu, Gang Niu, and Yang Liu. 2021. Learning with noisy labels revisited: A study using real-world human annotations. *arXiv preprint arXiv:2110.12088*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

Yu Zhang and Qiang Yang. 2021. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609.

Zhihan Zhang, Wenhao Yu, Mengxia Yu, Zhichun Guo, and Meng Jiang. 2022. A survey of multi-task learning in natural language processing: Regarding task relatedness and training methods. *arXiv preprint arXiv:2204.03508*.

A Algorithm to selectively apply attention to context vector for MTL-CNLU-SAWC

Algorithm 1 Selectively apply attention to context vector

if $\hat{y}_u \in$ Flow Intents List **then**
  $\tilde{c} \leftarrow a_i * c_i$
else
  $\tilde{c} \leftarrow c_i$
end if

B Algorithm to calculate top 2 score

Let y_u, y_c, $\hat{y}_1$ and $\hat{y}_2$ denote the utterance label, conversation label, 1st predicted intent and 2nd predicted intent respectively. For models with single

Algorithm 2 Top 2 score calculation

if $y_u = y_c$ **then**
  if $\hat{y}_1 = y_u$ or $\hat{y}_2 = y_u$ **then**
    score $\leftarrow 1$
  else
    score $\leftarrow 0$
  end if
else
  if $\hat{y}_1 \in \{y_u, y_c\}$ and $\hat{y}_2 \in \{y_u, y_c\}$ **then**
    score $\leftarrow 1$
  else if $\hat{y}_1 \in \{y_u, y_c\}$ or $\hat{y}_2 \in \{y_u, y_c\}$ **then**
    score $\leftarrow 0.5$
  else
    score $\leftarrow 0$
  end if
end if

head the 1st and 2nd predicted intents are the top 2 intent classes with maximum confidence. For models with two heads, the intent from the utterance head is considered as the 1st predicted intent and intent from the conversation head is considered to be the 2nd predicted intent.

C Ablation Study

A crucial element of our work involved pinpointing pertinent information from user transaction data to serve as context. To evaluate the significance of various context features, such as order level features and item level features, we compared the model's performance with and without these features. The findings from this analysis are presented in Table 4. Handcrafted features, such as "are any items left to be delivered" and "were any items cancelled" have the strongest influence on model accuracy. We utilized the CAWC architecture to make these comparisons.

Model trained on	top 1 accuracy (%)
text only (baseline)	78.65
text + order level features	79.34
text + item level features	78.91
text + order level + item level features	79.42
text + order level + item level features + handcrafted features	**81.04**

Table 4: Assessment of the contribution of different context features towards model performance improvement

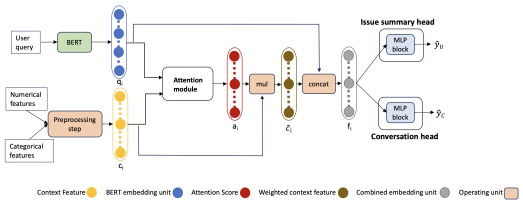

Figure 6: MTL-CNLU-shared: MTL-CNLU with query-context combining module shared between the heads

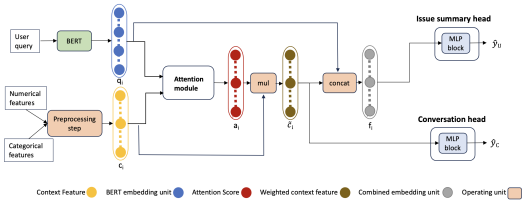

Figure 7: MTL-CNLU-AWC-shared: MTL-CNLU-AWC with query-context combining module shared between the heads

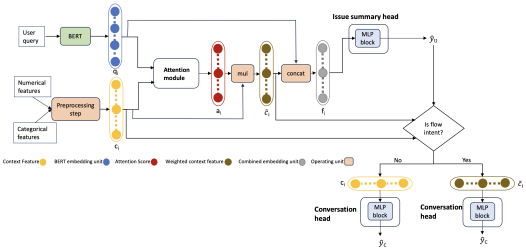

Figure 8: MTL-CNLU-SAWC-shared: MTL-CNLU-SAWC with query-context combining module shared between the heads

D Alternate architecture considerations for MTL-CNLU

As outlined in section 2.5, the core BERT module in all our MTL-CNLU-based architectures is

Architecture	Utterance Intent		Conversation Intent		Top 2 Score (%)
	Micro F1(%)	Macro F1(%)	Micro F1(%)	Macro F1(%)	
MTL-CNLU	81.65	78.80	38.65	37.80	89.90
MTL-CNLU-AWC	81.54	78.81	41.78	38.95	90.44
MTL-CNLU-SAWC	81.88	79.05	42.03	39.56	90.87
MTL-CNLU-shared	81.48	78.62	34.65	32.80	85.90
MTL-CNLU-AWC-shared	81.66	78.68	34.11	33.11	87.21
MTL-CNLU-SAWC-shared	81.62	78.81	34.77	32.56	87.18

Table 5: Results comparing performance of the two sets MTL-CNLU based architectures, ones where query-context combining module was also shared between the heads in addition to the BERT module and the ones where only the BERT module was shared

shared by both the utterance and conversation heads. However, each head possesses its own query-context combining module. We also experimented with using a single query-context combining module for both heads. This was carried out for each MTL-CNLU architecture, specifically MTL-CNLU, MTL-CNLU-AWC and MTL-CNLU-SAWC. The resulting new architectures are named MTL-CNLU-shared, MTL-CNLU-AWC-shared, and MTL-CNLU-SAWC-shared respectively. Detailed architectures are available in figures 6, 7, and 8. Each of these models demonstrated lower performance than their counterparts with separate query-context combining modules, as shown in Table 5.

E Training details

All the model architectures described above were trained using the same dataset, with the only difference being the training methodology for MTL-CNLU models due to the incorporation of conversation labels. The BERT model employed as an encoder in all our architectures is a pre-trained distilled variant, Small BERT (Turc et al. 2019), with a hidden state dimension of 768. All models in this paper were trained using the AdamW (Loshchilov and Hutter 2017) optimizer with a learning rate of 0.0001. A dropout of 0.5 was applied to the layers in the MLP block. The first hidden layer in the MLP block utilized the ReLU activation function, while the tanh activation function was employed after layers within the attention module. The choice of activation function was based on empirical results. As previously mentioned, for MTL-CNLU, conversation labels serve as a proxy for labels that should have ideally been assigned according to the user's contextual information. Thus, they do not impact training when context is absent. Consequently, the hyperparameter λ is set to zero for training examples without context data. For the remaining examples, we experimented with values $\{0.6, 0.8, 1, 1.2, 1.4\}$ for λ and ultimately set it to 1 based on the results. A batch size of 32 was employed during training. The Tensorflow library was used for implementation, and all models were trained on an Nvidia V100 GPU. Our final model, MTL-CNLU-SAWC, contains approximately 39 million trainable parameters and takes an average of 3 hours to train on an Nvidia V100 GPU.

F Handcrafted context features used by the model

Table 6 shows some of the handcrafted features that were created to help our model learn complex feature interactions.

Feature	Feature type
time since last delivered item	numerical
time since last shipped item	numerical
time since last cancelled item	numerical
are any items left to be delivered	categorical
are all items left to be delivered	categorical
are any items left to be shipped	categorical
are any items past expected delivery time	categorical
are all items past expected delivery time	categorical
were any items cancelled by store	categorical
were any items cancelled by customer	categorical

Table 6: Some handcrafted features used by our model

G Dataset statistics

Dataset detail	Statistics
#utterances in training	100K
#utterances in validation	2.5K
#utterances in test	2.5K
% of examples with context	70%
% of examples where utterance label $\neq$ conversation label	45%
#intents covered by utterance labels	59
#intents covered by conversation labels	35

Table 7: Dataset statistics