IMPROVING USER EXPERIENCE IN THE IMPLEMENTATION OF THE PUBLIC PROCUREMENT LAW IN THE REPUBLIC OF SERBIA THROUGH AN INTERACTIVE CHATBOT BASED ON AI TECHNOLOGY

Dejan Dodić
EDUKOM D.O.O. Vranje, Serbia, dodic@edukom.rs

Abstract: This scientific research paper presents the development of a neural network for a computer model that will be applied in the development of a chatbot web application based on a proprietary dataset. The aim of the chatbot is to provide virtual consulting support in the implementation of the Public Procurement Law in the Republic of Serbia. This AI tool is of crucial importance for both procurers and bidders, given the need for understanding and compliance with the law. By using the complete content of texts related to the law's implementation available on the internet, the model will train its neural networks based on an algorithm to provide answers to various questions.
The chatbot will be capable of answering questions related to the definitions of the Public Procurement Law ("What is...?"), procedural questions ("What to do in the following case...?"), and ways to overcome obstacles in implementing the law ("How to overcome the problem...?"). By tracking the conversation context and searching through texts, the model will provide meaningful responses.
This project has the potential to revolutionize the implementation of the Public Procurement Law in Serbia, considering the large number of officials and the lack of consultants who could provide accurate real-time answers and solutions. Currently, there are numerous decisions of the Republic Commission for the Protection of Rights that represent the practice of law implementation, and the chatbot would enable access to this information to ensure the correctness and transparency of public procurement procedures.
Therefore, the model can be expanded and adapted for application with other laws, provided that there is appropriate textual material for model training and tokenization, which would further enhance the accuracy of responses to questions regarding the implementation of the Public Procurement Law.
It is important to note that as the conceptual creator of this concept, I have personally submitted a request to the Institute for Artificial Intelligence - AI Institute Novi Sad, Serbia, to collaborate on realizing this idea.
Keywords: Chatbot, neural networks, Python, Public Procurement Law (ZJN), hyperparameters, modeling, BM25Okapi, GPT-2, NLP, wandb, result analysis, model training, virtual consultant, revolution in consulting services, dataset.

1. INTRODUCTION

The development of AI-powered chatbots has garnered significant attention in recent years. This research paper focuses on the development of a chatbot specifically designed to provide virtual consulting support in the application of the Public Procurement Law (ZJN) in Serbia. The chatbot utilizes its own dataset. The aim of this AI tool is to address the key need for understanding and aligning the needs of procurers and bidders with the ZJN and provide value in its application.
By leveraging extensive textual content available on the internet related to the application of ZJN, the chatbot model employs neural networks trained with specific algorithms to generate responses to various queries. The chatbot is designed to handle questions related to the definition of public procurement law, procedural inquiries, and ways to overcome obstacles during law implementation. By considering the context of the conversation and applying efficient text search algorithms, the chatbot provides meaningful and relevant answers.
This approach paves the way for the chatbot to become a versatile tool with a learning capability in the legal domain, adaptable to different legislations and regulatory frameworks.
In the subsequent sections, we will delve into the methodology, architecture, and evaluation of the chatbot, showcasing its potential impact on improving the understanding of ZJN in Serbia while highlighting prospects for future expansion into other legal domains.

2. OBJECTIVE OF THE RESEARCH

The aim of this research is to develop an AI chatbot based on the Transformers libraries that will provide virtual consulting support in the implementation of the Public Procurement Law in the Republic of Serbia. Transformers have become key tools in natural language processing (NLP) and have shown exceptional ability in generating meaningful and coherent responses to language queries.

As part of this research, we will utilize popular libraries such as Hugging Face's Transformers and PyTorch. These libraries provide a rich set of models based on the Transformer architecture, such as BERT, GPT, and T5. This model will be trained on our own dataset related to the application of the Public Procurement Law in the Serbian language.
During the model training process, we will take advantage of GPU acceleration to achieve efficient and fast training. We assume access to a Linux Ubuntu server with an NVIDIA GeForce graphics card that has at least 16GB VRAM. This enables parallel computation and speeds up the model training process.
Here is an example Python script for training the model based on the Transformers library:

```python
import wandb
from transformers import GPT2Tokenizer, GPT2LMHeadModel
import torch

# Inicijalizacija wandb
wandb.init(project='chatbot-training')

# Učitavanje i inicijalizacija modela
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')

# Definisanje trening podataka
train_data = [
    "Prvi trening primer.",
    "Drugi trening primer."
]

# Tokenizacija i kodiranje trening podataka
input_ids = tokenizer(train_data, padding=True, truncation=True, return_tensors='pt').input_ids

# Treniranje modela
model.train()
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)

for epoch in range(num_train_epochs):
    optimizer.zero_grad()
    outputs = model(input_ids, labels=input_ids)
    loss = outputs.loss
    loss.backward()
    optimizer.step()

    # Praćenje gubitka i epohe pomoću wandb
    wandb.log({"loss": loss.item(), "epoch": epoch})

# Evaluacija modela
eval_data = [
    "Primer za evaluaciju 1.",
    "Primer za evaluaciju 2."
]

input_ids_eval = tokenizer(eval_data, padding=True, truncation=True, return_tensors='pt').input_ids

model.eval()
with torch.no_grad():
    outputs_eval = model.generate(input_ids_eval)

decoded_outputs = tokenizer.batch_decode(outputs_eval, skip_special_tokens=True)
```

```
# Prijavljivanje metrike evaluacije pomoću wandb
wandb.log({"eval_metric": 0.85})  # Ovo je samo ilustrativni primer, treba zameniti sa stvarnim vrednostima

# Završetak wandb sesije
wandb.finish()
```

This script utilizes the Transformers libraries for model loading and initialization, data tokenization and encoding, model training with the AdamW optimizer, and model evaluation on the evaluation data. In the example, the GPT model was used, but you can adapt the script to use other models available in the Transformers libraries.
Considering the technical aspects of the AI chatbot based on the Transformers libraries, the goal of this research is to achieve high accuracy and coherence in the chatbot's responses to various queries related to the application of the Public Procurement Law (ZJN). The use of Transformers models enables the chatbot to better understand the context of questions and generate relevant and informative answers.
Furthermore, the research aim is to customize the chatbot for the specific needs of the legal domain, with a focus on the Public Procurement Law in the Republic of Serbia. This means that the chatbot will be trained on relevant textual materials related to the application of that law. Through training on a large dataset, the chatbot will gain deeper understanding of the law, rules, procedures, and challenges that arise in the public procurement process.
The results of this research are expected to significantly enhance the understanding and application of the Public Procurement Law in the Republic of Serbia. The chatbot will be available as an interactive tool that provides users with quick access to accurate information and guidelines regarding law implementation. This will be of great benefit to both procurers and suppliers in public procurement procedures.
Additionally, the results of this research can have broader implications for the application of other laws and regulations. This AI chatbot can be adapted for use in other legal domains, enabling quick access to accurate information and guidelines from different legal contexts. This would improve transparency, efficiency, and accessibility of legal support or practices in various contexts.
Additionally, as part of the goal of this research, we will integrate with the wandb.ai platform to track the learning progress through epochs, as well as the crucial factor of loss during model training. Wandb.ai provides advanced capabilities for real-time data visualization and analysis, enabling researchers to effectively monitor model performance during the training process. This integration allows for detailed monitoring and evaluation of the model's progress, enabling further improvement of the chatbot's performance.
It is expected that this additional tracking and analysis capability through the wandb.ai platform will contribute to a deeper understanding of the model training process, identification of areas that require improvement, and optimization of the chatbot's performance. Additionally, it allows researchers to transparently showcase research results and share them with the AI community.
Through this research, we strive to develop an advanced AI chatbot to support the application of the Public Procurement Law, while simultaneously leveraging the advantages of the wandb.ai platform for tracking and evaluating learning progress. The goal is to create an efficient tool that enables users to quickly access accurate information, improve understanding of the law, and provide relevant guidance in the public procurement process.
In the subsequent stages of the research, we will delve into the architecture of the chatbot, training methodology, performance evaluation, and explore possibilities for further extending the chatbot to other areas of law. The aim is to create a versatile tool that can contribute to enhancing various aspects of the legal profession and facilitate access to accurate legal information.

3. RESEARCH METHODOLOGY

To achieve our goal and develop the AI chatbot, we will follow the following methodology:
Data selection and preparation: For training the AI chatbot, we use our own dataset related to the application of the Public Procurement Law in the Serbian language. These data include relevant textual materials, laws, regulations, resolutions, tender documentation, rules, and procedures related to public procurement. The data is collected, cleaned, and transformed into a format suitable for model training.
Model selection and configuration: We base our chatbot on the GPT model from the Transformers library. The GPT model is known for its exceptional performance in generating coherent responses to language queries. With the GPT model, we can better understand the context of questions and generate relevant and meaningful responses. We customize the GPT model to our specific needs and configure it for training.

Model training: We use a Python script for training the model based on the Transformers library. The script runs on a Linux Ubuntu server with GPU acceleration. We train the model using the AdamW optimizer and minimize the loss during the training process. Additionally, we utilize the wandb.ai library to track the loss and epochs during training for detailed analysis and evaluation of the model's progress.
Model evaluation: After training, we evaluate the model using evaluation data that contains various examples of queries related to the Public Procurement Law. We generate responses based on the evaluation data and measure the quality of the generated answers. We use an evaluation metric, **eval_metric**, to quantify the model's performance. Wandb.ai enables us to track and display these metrics in real-time.
Integration with the wandb.ai platform: As an additional step, we have integrated our model training script with the wandb.ai platform to track the learning progress through epochs, loss, and performance evaluation. Wandb.ai provides advanced capabilities for real-time data visualization and analysis, making it easier for researchers to monitor model performance during the training process.

```
import wandb
from transformers import GPT2Tokenizer, GPT2LMHeadModel
import torch

# Hiperparametri za treniranje modela
per_device_train_batch_size = 4  # Broj uzoraka (instance) po uređaju
num_train_epochs = 5  # Broj epoha treniranja
learning_rate = 3e-5  # Brzina učenja modela
weight_decay = 0.1  # Parametar koji pomaže u sprečavanju prenaučenosti modela
warmup_steps = 200  # Broj koraka treniranja za postepeno povećavanje brzine učenja
logging_steps = 200  # Broj koraka nakon kojih se izveštava o napretku modela
max_length = 512  # Maksimalna dužina ulaza u model

# Inicijalizacija wandb
wandb.init(project='chatbot-training')

# Učitavanje i inicijalizacija modela
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')

# Definisanje trening podataka
train_data = [
    "Prvi trening primer.",
    "Drugi trening primer."
]

# Tokenizacija i kodiranje trening podataka
input_ids = tokenizer(train_data, padding=True, truncation=True, return_tensors='pt').input_ids

# Treniranje modela
model.train()
optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate, weight_decay=weight_decay)

total_steps = len(input_ids) * num_train_epochs // per_device_train_batch_size
warmup_steps = min(warmup_steps, total_steps)

scheduler = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr=learning_rate,
                                                total_steps=total_steps, epochs=num_train_epochs,
                                                steps_per_epoch=len(input_ids) // per_device_train_batch_size,
                                                anneal_strategy='linear', warmup_steps=warmup_steps)

for epoch in range(num_train_epochs):
    optimizer.zero_grad()
    outputs = model(input_ids, labels=input_ids)
    loss = outputs.loss
    loss.backward()
    optimizer.step()
    scheduler.step()

    # Praćenje gubitka i epohe pomoću wandb
    wandb.log({"loss": loss.item(), "epoch": epoch})

# Evaluacija modela
eval_data = [
    "Primer za evaluaciju 1.",
    "Primer za evaluaciju 2."
]

input_ids_eval = tokenizer(eval_data, padding=True, truncation=True, return_tensors='pt').input_ids

model.eval()
with torch.no_grad():
    outputs_eval = model.generate(input_ids_eval)

decoded_outputs = tokenizer.batch_decode(outputs_eval, skip_special_tokens=True)

# Prijavljivanje metrike evaluacije pomoću wandb
wandb.log({"eval_metric": 0.85})  # Ovo je samo ilustrativni primer, treba zameniti sa stvarnim vrednostima

# Završetak wandb sesije
wandb.finish()
```

Hyperparameters are parameters that are set before the actual model training process and influence the way the model is trained. They differ from model parameters that are learned during training. Hyperparameters are used to adjust various aspects of the training process and can have a significant impact on the model's performance and behavior. Their values are chosen experimentally or based on previous experience.
Here is a detailed explanation of the additional hyperparameters in our experimental script:
per_device_train_batch_size: This hyperparameter defines the number of samples (instances) processed in parallel on each device during one training iteration. A larger per_device_train_batch_size can speed up the training process but may also require more memory on the device. It is important to adjust this parameter based on available resources and the size of the training dataset.
num_train_epochs: This hyperparameter represents the number of epochs, i.e., the number of times the entire training dataset will pass through the model. Increasing the number of epochs can help the model learn more complex patterns but may also lead to overfitting. It is important to find an appropriate number of epochs that leads to optimal results.
learning_rate: This hyperparameter defines the learning rate of the model, i.e., how quickly the model adapts its weights during training. A lower value can result in more precise learning but may prolong the training time, while a higher value can speed up training but may skip optimal points. Choosing the right learning rate is crucial for achieving good model performance.
weight_decay: This hyperparameter is used to prevent model overfitting by adding a small additional term to the weights during training. It helps reduce the influence of irrelevant or harmful weights. The value of weight_decay should be carefully chosen to achieve a proper balance between generalization and fitting to the training data.
warmup_steps: This hyperparameter defines the number of training steps during which the learning rate gradually increases. This technique, known as "warmup," helps the model adapt gradually and avoids abrupt jumps in weights that can lead to instability in learning. Properly adjusting this parameter can help improve model convergence.
logging_steps: This hyperparameter defines the number of steps after which information about the model's progress during training will be logged. Reporting on the model's progress can be useful for monitoring performance during training and identifying potential issues. For example, you can track the loss over time to check if the model is

converging or if overfitting occurs. You can also track evaluation metrics to gain insight into the model's performance during training.

max_length: This hyperparameter represents the maximum length of input sequences that the model will process. If an input sequence is longer than max_length, it will be truncated or cut to match the specified length. Limiting the length of input sequences can be useful for efficiency reasons as shorter sequences require fewer resources for processing. However, care should be taken not to lose too much information with truncated or cut sequences.

These additional hyperparameters provide the ability to fine-tune the model training process. Adapting these parameters based on the specific project requirements, available resources, and data characteristics can help achieve better results and more efficient training of an AI chatbot based on the Transformers library.

4. RESEARCH RESULTS

After completing the training of the model on our custom dataset, we will proceed to testing the model. In our research work, we used a script that allows us to process the research results. The script utilizes the GPT-2 language model to generate responses to the given text based on the posed questions.

```
import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer
from rank_bm25 import BM25Okapi
import nltk
import transformers
from transformers import GPT2TokenizerFast

nltk.download("punkt")

model_path = "/putanja/do/modela/model"
text_path = "/putanja/do/teksta/tekst.txt"

# Učitavanje modela i tokenizera
model = GPT2LMHeadModel.from_pretrained(model_path)

# Instanciranje tokenizera
tokenizer = GPT2TokenizerFast.from_pretrained(model_path)

# Postavljanje pad_token
tokenizer.pad_token = tokenizer.eos_token

# Učitavanje teksta
with open(text_path, "r") as file:
    text = file.read()

def generate_answer_with_bm25(question, text):
    text_sentences = nltk.sent_tokenize(text)

    tokenized_text = [nltk.word_tokenize(sentence) for sentence in text_sentences]
    bm25 = BM25Okapi(tokenized_text)

    tokenized_question = nltk.word_tokenize(question)
    scores = bm25.get_scores(tokenized_question)
    top_n_sentences = sorted(range(len(scores)), key=lambda i: scores[i], reverse=True)[:5]

    relevant_sentences = [text_sentences[index] for index in top_n_sentences]
    print("Relevant sentences:", relevant_sentences)  # Dodavanjem ovog ispisa možemo prikazati relevantne rečenice
    return relevant_sentences

def get_answer(question, context):
    input_text = f"{question.strip()} {context.strip()}"
    print(f"Input text: '{input_text}'")
    encoded_input = tokenizer(input_text, return_tensors="pt", truncation=True, padding=True, max_length=512)
    generated_ids = model.generate(encoded_input['input_ids'], attention_mask=encoded_input['attention_mask'],
max_length=512, num_return_sequences=1, no_repeat_ngram_size=2, do_sample=True, temperature=0.7)

    answer = tokenizer.decode(generated_ids[0], skip_special_tokens=True)
    return answer.split(question)[-1].strip()

# Unos pitanja putem komandne linije u terminalu
question = input("\n\nUnesite svoje pitanje: ")

# Generisanje najrelevantnije rečenice pomoću BM25 algoritma
context_sentences = generate_answer_with_bm25(question, text)

# Spojanje rečenica u jedan tekst
context = ' '.join(context_sentences)

# Generisanje odgovora koristeći GPT-2 model
answer = get_answer(question, context)
print("\n\n", answer)
```

Here are the details of each part of the script and their roles.
Loading the model and tokenizer: The script first loads the pre-trained GPT-2 model from the specified model_path using the GPT2LMHeadModel.from_pretrained() function. Then, it instantiates the tokenizer using the GPT2TokenizerFast.from_pretrained() function, which is used to tokenize the input text.
Loading the text: The script loads the text from the specified text file path given in the text_path variable using the open() function and the read() method. The loaded text is stored in the text variable and used as the context for generating responses.
generate_answer_with_bm25(question, text) function: This function utilizes the BM25Okapi algorithm to rank relevant sentences in the text based on the given question. First, the text is split into sentences using the nltk.sent_tokenize() function, and then each sentence is tokenized into words. Then, the BM25Okapi algorithm from the rank_bm25 library is used to calculate the relevance scores between the posed question and each sentence. The top five most relevant sentences are selected based on these scores, processed, and stored in the relevant_sentences variable. The relevant sentences are displayed using the print("Relevant sentences:", relevant_sentences) statement.
get_answer(question, context) function: This function uses the GPT-2 model to generate an answer to the given question using the provided context. The input text is formed by concatenating the question and the context. Tokenization and encoding of the input text are performed using the tokenizer object. Then, the model.generate() function is used to generate a response based on the encoded input. The generated response is decoded into readable form using the tokenizer.decode() function. The answer is obtained by extracting the part of the generated text that comes after the question.
Question input and answer generation: The script allows the user to input questions via the command line in the terminal. After the question is entered, the generated answer is displayed on the standard output.
This script enables automatic generation of answers to posed questions based on the given text. The combination of using the BM25Okapi algorithm for identifying relevant sentences and the GPT-2 model for generating responses enables efficient and adaptable text research and analysis within our work.

5. CONCLUSION

In conclusion, we can observe that the research was focused on various aspects of scientific research, particularly emphasizing research methodology, the use of hyperparameters in modeling, and analysis of research results. Through different examples and discussions, we have gained a deeper understanding of this concept and its role in the scientific research process.
We have considered the significance of research methodology, which enables a systematic approach to data collection, analysis, and interpretation. The importance of hyperparameters in modeling and their role in optimization and model tuning is one of the key factors for the success of a model. Through examples, we have

explained how hyperparameters affect the performance of a model. We have understood that proper tuning of hyperparameters can enhance the efficiency, accuracy, and generalization of the model, while inappropriate hyperparameters can lead to overfitting or inadequate accuracy. We have also recognized the importance of critical analysis of results, tracking evaluation metrics, and their interpretation.
Overall, the combination of a rigorous research approach, proper model tuning, and thorough analysis of results forms the foundation for the successful development of a chatbot based on our own dataset, aiming to provide virtual consultancy support in the implementation of the Public Procurement Law in the Republic of Serbia.

LITERATURE

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. Proceedings of the 26th Annual International Conference on Machine Learning (ICML), 41, 41-48. https://doi.org/10.1145/1553374.1553380
Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(Feb), 281-305. http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf
Brown, T. B., et al. (2020). Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems 33 (NeurIPS 2020).
Brown, T. B., et al. (2020). Language Models are Few-Shot Learners. arXiv preprint arXiv:2005.14165.
Dai, Z., et al. (2019). Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019).
Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. https://arxiv.org/abs/1412.6980
Liu, Y., et al. (2020). Generative Pre-trained Transformer 3 (GPT-3). arXiv preprint arXiv:2005.14165.
Liu, Y., et al. (2020). On the Variance of the Adaptive Learning Rate and Beyond. Proceedings of the 7th International Conference on Learning Representations (ICLR 2019).
Radford, A., et al. (2019). Language Models are Unsupervised Multitask Learners. OpenAI Blog.
Radford, A., et al. (2019). Language Models are Unsupervised Multitask Learners. arXiv preprint arXiv:1910.10683.
Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI Blog, 1(8). https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
Raffel, C., et al. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arXiv preprint arXiv:1910.10683.
Sasaki, Y. (2007). The truth of the F-measure. The University of Tokyo. http://acl.ldc.upenn.edu/W/W04/W04-3258.pdf
Vaswani, A., et al. (2017). Attention Is All You Need. Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS 2017).
Wolf, T., et al. (2020). Transformers: State-of-the-Art Natural Language Processing. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020).