Requirements Recognition for Verification IP Design Using Large Language Models

S. S. Zalivaka
SK hynix NAND Product Solutions Poland
224 Juliusza Slowackiego,
Gdansk, 80-298

***Abstract*-Verification becomes more crucial in the design of complex systems on a chip. To make this process more sustainable, the verification IPs (VIPs) have to be utilized. The internals of the VIP are usually designed based on the requirements, which cannot be easily extracted from the specification. Thus, Large Language Models (LLMs) have been used in order to automatically identify the requirements. The experimental study shows that GPT-2 model provides good performance (Precision=0.618, Recall=0.874) on the data extracted from M-PHY 4.1 specification. The trained model has shown better performance comparing to the generally trained bigger model (GPT 3.5). The proposed model has also been integrated into an automation system for technical documents analysis. The system has shown the improvement of the requirements extraction process by at least 20% compared to the traditional manual process.**

I. Introduction

Nowadays, modern systems on a chip (SoC) become more sophisticated due to high complexity of their design. As a result, a typical SoC design includes many external reusable Intellectual Property (IP) cores, which drastically reduce the design time. On the other hand, verification process may take more than half of the total design time. Therefore, in order to reduce verification time, engineers utilize verification IPs (VIPs) [1].

According to the definition, verification process is aiming to check whether the system meets requirements described in the specification [2]. Thus, any verification process requires deep understanding of the system specification. Engineers decompose specifications into a set of requirements to develop test benches, stimulus sequences, functional models, checkers, coverage model, etc.

Existing specifications for widely used IPs (e.g., PCIe [3]) may have an extraordinary volume, i.e., thousands of pages. This makes manual processing of frequently changing specifications a task requiring a high amount of time (days or months) even for an expert in this field. Thus, the Large Language Models (LLM) are proposed to reduce the amount of time required for manual specification analysis.

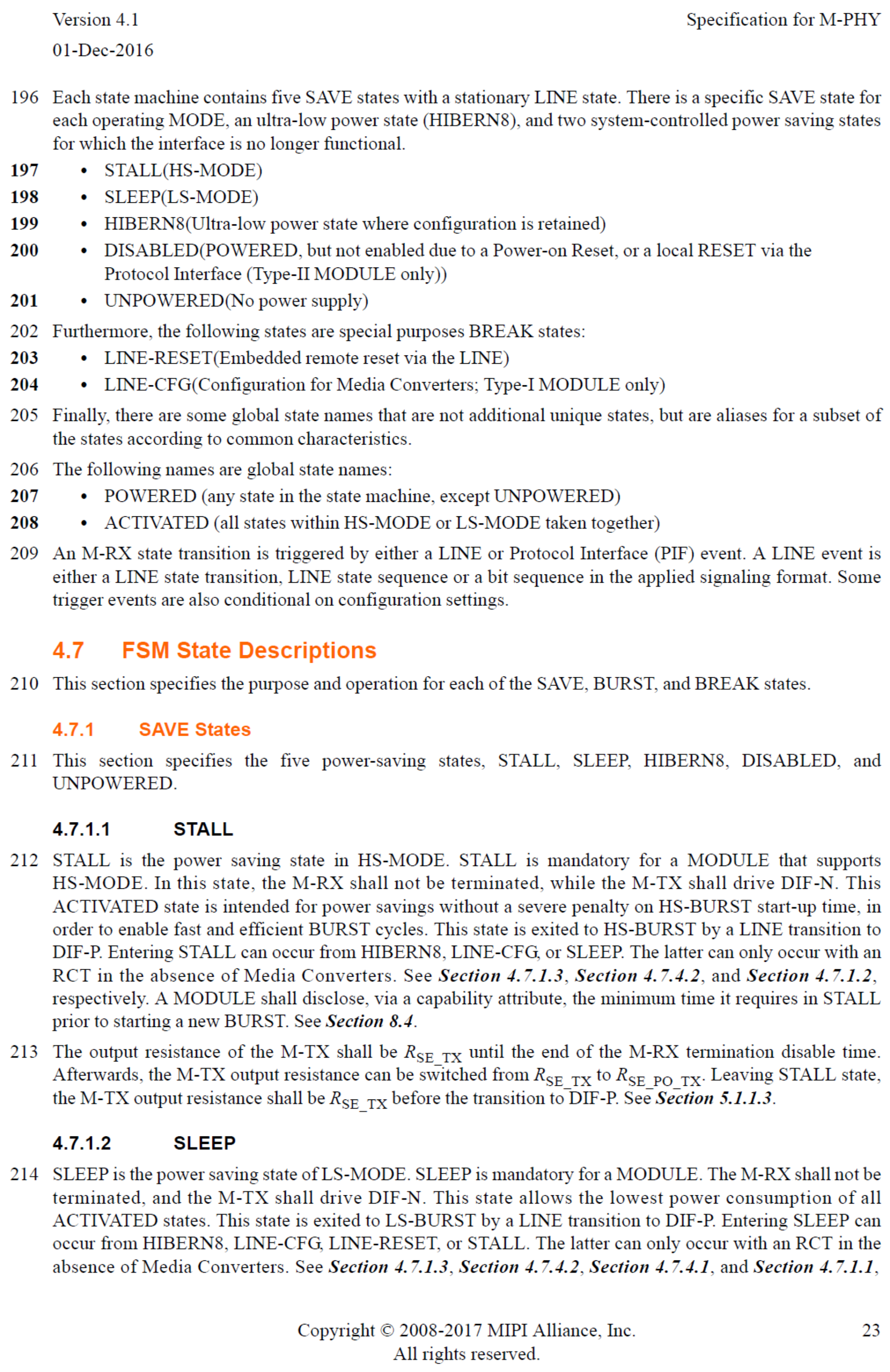
The experiment has been conducted on the specification of M-PHY v. 4.1 [4], which has been manually analyzed, parsed and labeled by a group of engineers. This dataset was used to fine-tune Generative Pre-trained Transformer 2 (GPT-2) model from Hugging Face library [5]. The performance of the model has been also assessed on PURE dataset [6] to show that more specific data is more suitable for training a model. The proposed approach has been compared to generally trained GPT 3.5 model and has shown better ability to recognize the requirements. The trained model has been utilized as a part of automation system for technical documents analysis, which has been tested by labeling PCIe specification [3].

II. Prior Works

Since requirements engineering is one of the important steps of the technical systems development process, the idea of automating this time-consuming process is straightforward. The early application of natural language processing (NLP) methods has established in early 90-s [7]. The further advancement of these methods went side by side with advancement of machine learning (ML) algorithms and datasets development. Nowadays, there are three most popular datasets in this area, i.e., PROMISE [8], DOORS [9] and PURE [6] and the range of used ML techniques varies from the linear models [10] to the advanced deep learning methods [11]. The performance of the most successful methods on either precision or recall can achieve 0.90 [12]. However, the industrial success of these methods relies on domain-specific rather than on a general-purpose datasets [13].

III. Datasets Description

The in-house VIP for M-PHY 4.1 has been developed in order to improve the verification process. As a first stage of the development, the specification has been manually analyzed by a group of engineers in order to automate protocol verification, provide functional coverage, add error injection, and develop customizable test benches. All mentioned functions are based on the requirements, which have been extracted from the specification.



196 Each state machine contains five SAVE states with a stationary LINE state. There is a specific SAVE state for each operating MODE, an ultra-low power state (HIBERN8), and two system-controlled power saving states for which the interface is no longer functional.

197 • STALL(HS-MODE)
198 • SLEEP(LS-MODE)
199 • HIBERN8(Ultra-low power state where configuration is retained)
200 • DISABLED(POWERED, but not enabled due to a Power-on Reset, or a local RESET via the Protocol Interface (Type-II MODULE only))
201 • UNPOWERED(No power supply)

202 Furthermore, the following states are special purposes BREAK states:

203 • LINE-RESET(Embedded remote reset via the LINE)
204 • LINE-CFG(Configuration for Media Converters; Type-I MODULE only)

205 Finally, there are some global state names that are not additional unique states, but are aliases for a subset of the states according to common characteristics.

206 The following names are global state names:

207 • POWERED (any state in the state machine, except UNPOWERED)
208 • ACTIVATED (all states within HS-MODE or LS-MODE taken together)

209 An M-RX state transition is triggered by either a LINE or Protocol Interface (PIF) event. A LINE event is either a LINE state transition, LINE state sequence or a bit sequence in the applied signaling format. Some trigger events are also conditional on configuration settings.

4.7 FSM State Descriptions

210 This section specifies the purpose and operation for each of the SAVE, BURST, and BREAK states.

4.7.1 SAVE States

211 This section specifies the five power-saving states, STALL, SLEEP, HIBERN8, DISABLED, and UNPOWERED.

4.7.1.1 STALL

212 STALL is the power saving state in HS-MODE. STALL is mandatory for a MODULE that supports HS-MODE. In this state, the M-RX shall not be terminated, while the M-TX shall drive DIF-N. This ACTIVATED state is intended for power savings without a severe penalty on HS-BURST start-up time, in order to enable fast and efficient BURST cycles. This state is exited to HS-BURST by a LINE transition to DIF-P. Entering STALL can occur from HIBERN8, LINE-CFG, or SLEEP. The latter can only occur with an RCT in the absence of Media Converters. See ***Section 4.7.1.3***, ***Section 4.7.4.2***, and ***Section 4.7.1.2***, respectively. A MODULE shall disclose, via a capability attribute, the minimum time it requires in STALL prior to starting a new BURST. See ***Section 8.4***.

213 The output resistance of the M-TX shall be R_{SE_TX} until the end of the M-RX termination disable time. Afterwards, the M-TX output resistance can be switched from R_{SE_TX} to $R_{SE\_PO\_TX}$. Leaving STALL state, the M-TX output resistance shall be R_{SE_TX} before the transition to DIF-P. See ***Section 5.1.1.3***.

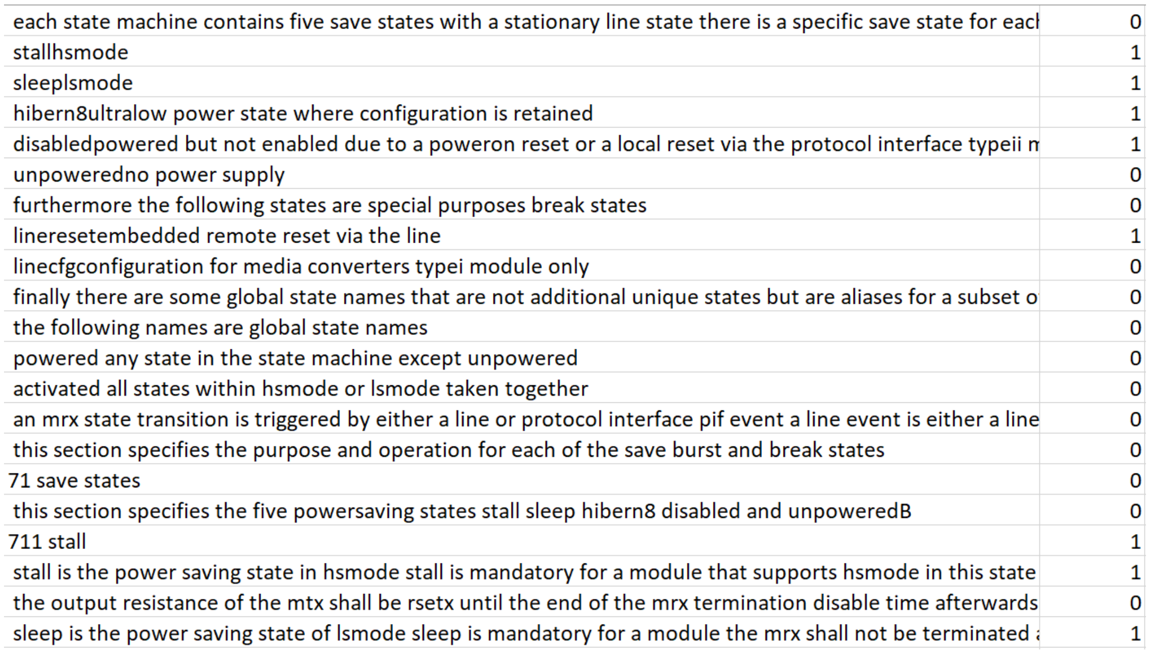
4.7.1.2 SLEEP

214 SLEEP is the power saving state of LS-MODE. SLEEP is mandatory for a MODULE. The M-RX shall not be terminated, and the M-TX shall drive DIF-N. This state allows the lowest power consumption of all ACTIVATED states. This state is exited to LS-BURST by a LINE transition to DIF-P. Entering SLEEP can occur from HIBERN8, LINE-CFG, LINE-RESET, or STALL. The latter can only occur with an RCT in the absence of Media Converters. See ***Section 4.7.1.3***, ***Section 4.7.4.2***, ***Section 4.7.4.1***, and ***Section 4.7.1.1***,

Figure 1. The page 23 of MPHY 4.1 specification divided into text fragments and numbered from 196 to 214.

The M-PHY v. 4.1 specification consists of 1130 text fragments, which have already been divided into sections in the specification file. The example of such text fragments is shown in Fig. 1. Based on manual analysis of specification contents, the engineers have extracted 332 requirements, the other 798 text fragments have been marked as non-requirements.

As a result, these text fragments have been processed, i.e., lowercased and the punctuation symbols have been removed. The part of the dataset, corresponding to the part of specification in Fig. 1, is shown in Fig. 2. On the other hand, there is a publicly available PURE dataset [6], which contains bigger number of text fragments (4145 are requirements and 3600 are non-requirements, 7745 in total). This dataset has more diverse data and a bigger number of records, but it is less specific and less suitable for the requirements recognition for the VIP design compared to the manually collected data from the hardware protocol specification.



each state machine contains five save states with a stationary line state there is a specific save state for eacl	0
stallhsmode	1
sleeplsmode	1
hibern8ultralow power state where configuration is retained	1
disabledpowered but not enabled due to a poweron reset or a local reset via the protocol interface typeii m	1
unpoweredno power supply	0
furthermore the following states are special purposes break states	0
lineresetembedded remote reset via the line	1
linecfgconfiguration for media converters typei module only	0
finally there are some global state names that are not additional unique states but are aliases for a subset o	0
the following names are global state names	0
powered any state in the state machine except unpowered	0
activated all states within hsmode or lsmode taken together	0
an mrx state transition is triggered by either a line or protocol interface pif event a line event is either a line	0
this section specifies the purpose and operation for each of the save burst and break states	0
71 save states	0
this section specifies the five powersaving states stall sleep hibern8 disabled and unpoweredB	0
711 stall	1
stall is the power saving state in hsmode stall is mandatory for a module that supports hsmode in this state	1
the output resistance of the mtx shall be rsetx until the end of the mrx termination disable time afterwards	0
sleep is the power saving state of lsmode sleep is mandatory for a module the mrx shall not be terminated	1

Figure 2. The part of the dataset corresponding to the page 23 of the M-PHY 4.1 specification.

The quality of these datasets can be experimentally determined based on the machine learning model performance on a validation dataset. In this case, the validation dataset is also based on the M-PHY 4.1 specification data. The difference between the M-PHY 4.1 based training and validation datasets is in the way of parsing it into text fragments. The training dataset is parsed manually into big fragments of text, i.e., one or several paragraphs. The validation dataset is parsed in an automatic way into sentences in order to differ from the training dataset.

The validation dataset also may include different kinds of parsing inaccuracies (e.g., additional symbols, words with missed/wrongly parsed letters, parts of the sentence located on the different pages and considered as two different sentences, etc.). As a result, despite containing approximately the same text content, the data significantly differs. In addition, the validation set is chosen in this way in order to be more closely aligned with the VIP design process rather than a general requirements recognition task.

III. Large Language Models

Large Language Models (LLM) [14] are machine learning models aiming understanding and generation of natural language. This can be achieved by using significant amounts of data in order to tune billions of internal parameters. These models can solve different natural language processing (NLP) tasks such as text classification, machine translation, question answering, automatic summarization, named-entity recognition, etc. [15]. One of the main LLMs nowadays are developed by Google (BERT and its modifications) [16], OpenAI (GPT-2, GPT-3, GPT-4) [17], Facebook (LLaMA, LLaMA2) [18]. An experimental study [12] has shown that the result of the requirements recognition task depends more on the quality of the data rather than on the used type of the LLM. The experiment carried out for this paper has also shown that the difference between BERT, GPT-2 and LLaMa models on the same dataset described above is not significant. Since GPT-2 model has shown a slightly better performance, the further experiments have been carried out with this model.

GPT models are based on the similar structure, which is shown in Fig. 3. All GPT models include Embedding block, which encodes the words into vector of numbers, and Positional Encoding block, which considers the order of words in a text sequence. Dropout block is used to reduce the overfitting, i.e., the situation when the model performs well on training data and poorly on validation data. The main block of GPT models is the Transformer Layer, which is implemented based on two main ideas, i.e., classical multilayer perceptron and attention helping to recognize dependencies between words in the fragment of text.

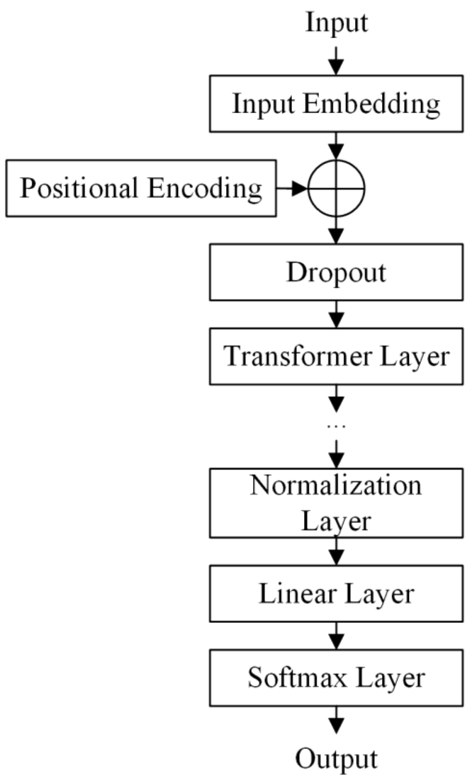


Figure 3. General structure of a GPT model.

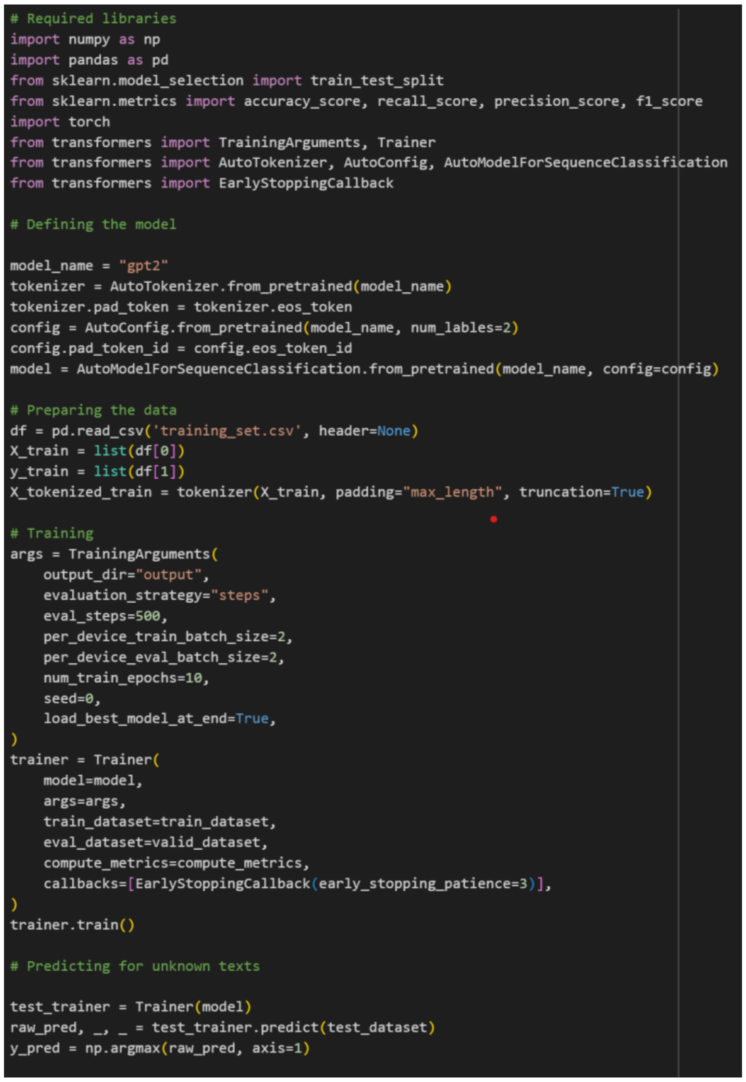
Normalization Layer is used to reduce the values of the weights in order to encode the weights with the smaller bit values (e.g., float16 instead of float32). Linear and Softmax layers form a classifier, which outputs the probabilities of being the next word corresponding to the words from the vocabulary.

The difference between GPT models is mainly in the number of Transformer Layers, the number of parameters and the amount of data used for the training process. Unfortunately, pre-trained models newer than GPT-2 are not openly available and full training on hundreds of GBs of training data is a time and resource consuming process. Thus, GPT-2 model is openly available and big enough to show good results on the considered requirements recognition task.

IV. Model Training Process

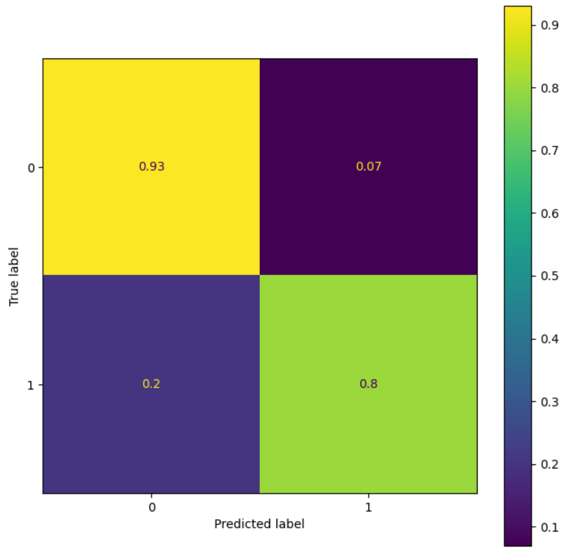
The initial GPT-2 has been modified in order to adapt the model to the requirement classification task, i.e., binary classification task (class ‘0’ is a non-requirement and class ‘1’ is a requirement). Since the classes are imbalanced, accuracy metrics may be misleading. Therefore, the model performance has been assessed using Precision and Recall metrics. Precision metric shows the percentage of correctly classified text fragments among the found ones, i.e., if the model classifies a piece of text as a requirement, Precision characterizes the certainty in the correct result. Recall metric shows the percentage of correctly found requirements among the whole set of requirements, i.e., the model with high recall likely found all the requirements but can classify some of the non-requirements wrongly. The modified GPT-2 model has been trained using GPUs on both M-PHY and PURE datasets. The training has been performed as a fine-tuning, i.e., that most parameters have been frozen (untrainable) and only the weights in a few last layers have been updated within the training process. The model is entirely implemented in-house and does not depend on any external cloud services.

The main parts of the source code for training are shown in Fig. 4.

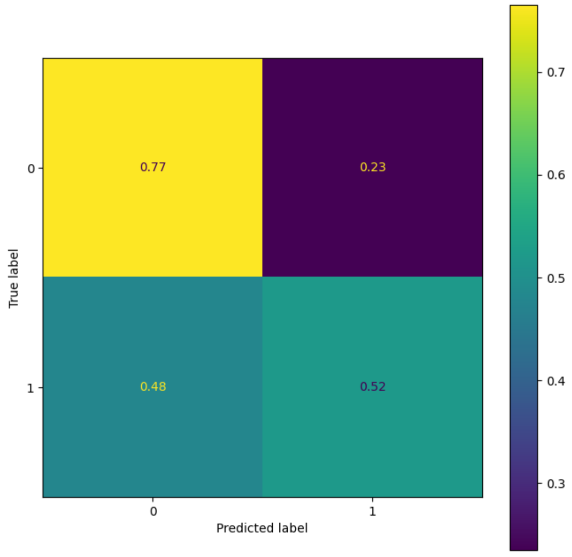


```python
# Required libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
import torch
from transformers import TrainingArguments, Trainer
from transformers import AutoTokenizer, AutoConfig, AutoModelForSequenceClassification
from transformers import EarlyStoppingCallback

# Defining the model

model_name = "gpt2"
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token
config = AutoConfig.from_pretrained(model_name, num_lables=2)
config.pad_token_id = config.eos_token_id
model = AutoModelForSequenceClassification.from_pretrained(model_name, config=config)

# Preparing the data
df = pd.read_csv('training_set.csv', header=None)
X_train = list(df[0])
y_train = list(df[1])
X_tokenized_train = tokenizer(X_train, padding="max_length", truncation=True)

# Training
args = TrainingArguments(
    output_dir="output",
    evaluation_strategy="steps",
    eval_steps=500,
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    num_train_epochs=10,
    seed=0,
    load_best_model_at_end=True,
)
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    compute_metrics=compute_metrics,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=3)],
)
trainer.train()

# Predicting for unknown texts

test_trainer = Trainer(model)
raw_pred, _, _ = test_trainer.predict(test_dataset)
y_pred = np.argmax(raw_pred, axis=1)
```

Figure 4. Main parts of the source code in Python for the model implementation.



(a)



(b)

Figure 5. Confusion matrices for M-PHY (a) and PURE (b) datasets.

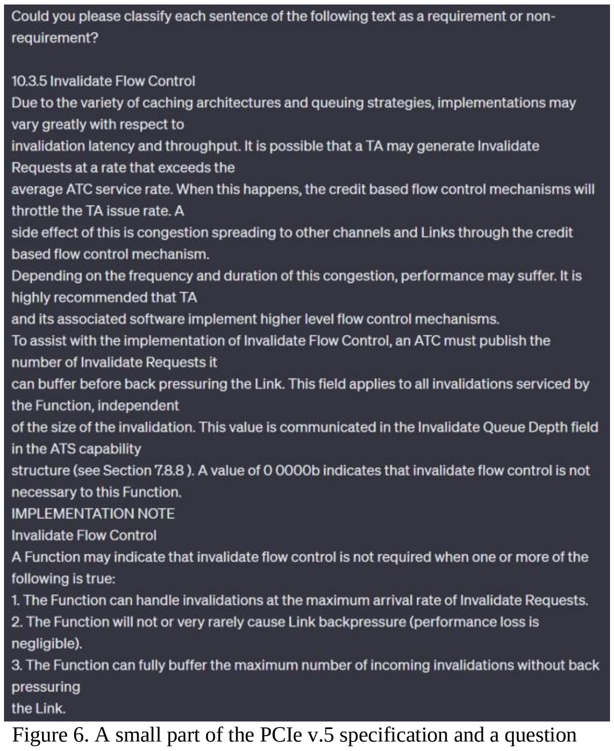
As a result, training time for this dataset is not significant (40-50 minutes). The accuracy on a training set has reached 0.995 and 0.873 on a validation set for M-PHY dataset. For the PURE dataset, the accuracy values on a training and validation sets are 0.982 and 0.659 correspondingly. The Precision and Recall on a validation set for M-PHY dataset have reached 0.618 and 0.874, correspondingly. For the PURE dataset, the Precision and Recall values are 0.513 and 0.778, correspondingly. The confusion matrices for the M-PHY and PURE datasets are shown in Fig. 5. The confusion matrix shows the percentage (or actual number) of correct and incorrect classifications for all classes. For the binary classification, the (0, 0) cell shows the percentage of correct classifications for the class 0, the (0, 1) cell shows the percentage of incorrect classifications for the class 0, the (1, 0) cell shows the percentage of incorrect classifications for the class 1, the (1, 1) cell shows the percentage of correct classifications for the class 1.

Comparing the results, the performance of the model is better on M-PHY specification-based dataset. This can be explained by a better specificity of this dataset compared to the more general PURE dataset. Since the accuracy on a validation set is significantly lower than on a training set, the model suffers from overfitting, i.e., the training set is not enough in terms of quantity and diversity to get a superior performance on different kinds of specifications. However, the model makes easier the process of labeling the requirements in the unknown specification, as it gives inputs for a quicker decision-making. Since the amount of data is not that big and the model is fine-tuned, the LLM does not suffer from inaccuracy and hallucinations.

IV. Model Assessment

A. Comparison to ChatGPT

The model has also been compared to the GPT 3.5 model [19], openly available as an API. The text (small part of PCIe-v.5 specification) shown in Fig. 6 has been processed in order to extract requirements using both the proposed model and the GPT 3.5 model (ChatGPT implementation).

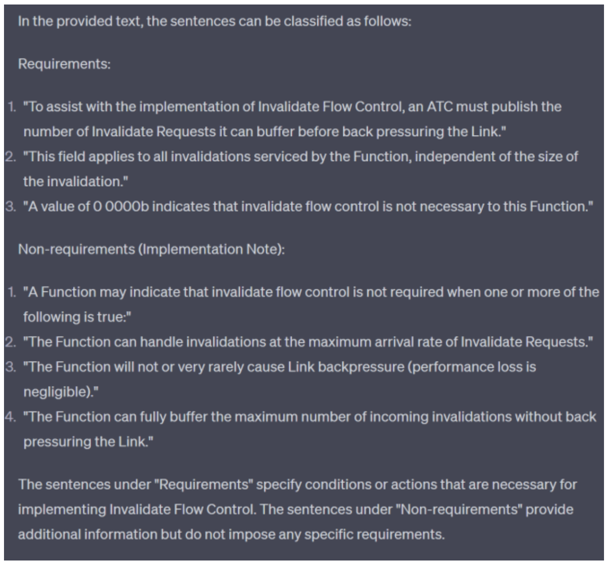

Could you please classify each sentence of the following text as a requirement or non-requirement?

10.3.5 Invalidate Flow Control
Due to the variety of caching architectures and queuing strategies, implementations may vary greatly with respect to
invalidation latency and throughput. It is possible that a TA may generate Invalidate Requests at a rate that exceeds the
average ATC service rate. When this happens, the credit based flow control mechanisms will throttle the TA issue rate. A
side effect of this is congestion spreading to other channels and Links through the credit based flow control mechanism.
Depending on the frequency and duration of this congestion, performance may suffer. It is highly recommended that TA
and its associated software implement higher level flow control mechanisms.
To assist with the implementation of Invalidate Flow Control, an ATC must publish the number of Invalidate Requests it
can buffer before back pressuring the Link. This field applies to all invalidations serviced by the Function, independent
of the size of the invalidation. This value is communicated in the Invalidate Queue Depth field in the ATS capability
structure (see Section 7.8.8). A value of 0 0000b indicates that invalidate flow control is not necessary to this Function.
IMPLEMENTATION NOTE
Invalidate Flow Control
A Function may indicate that invalidate flow control is not required when one or more of the following is true:
1. The Function can handle invalidations at the maximum arrival rate of Invalidate Requests.
2. The Function will not or very rarely cause Link backpressure (performance loss is negligible).
3. The Function can fully buffer the maximum number of incoming invalidations without back pressuring
the Link.

Figure 6. A small part of the PCIe v.5 specification and a question

The GPT 3.5 model has generated an output shown in Fig. 7.

The GPT 3.5 has ignored the sentences between "Due to the variety ..." and "To assist with ..." and has not marked them as "non-requirements". Also, the non-requirements 2, 3, 4 can be considered as requirements, which have been detected by the proposed model. As a result, the smaller model (GPT-2 in the proposed technique) completed this task better than the bigger model (GPT 3.5). Since the proposed model has been fine-tuned on the specific data related to the specifications, it shows better performance compared to the LLM trained on a general text data.


In the provided text, the sentences can be classified as follows:

Requirements:

1. "To assist with the implementation of Invalidate Flow Control, an ATC must publish the number of Invalidate Requests it can buffer before back pressuring the Link."
2. "This field applies to all invalidations serviced by the Function, independent of the size of the invalidation."
3. "A value of 0 0000b indicates that invalidate flow control is not necessary to this Function."

Non-requirements (Implementation Note):

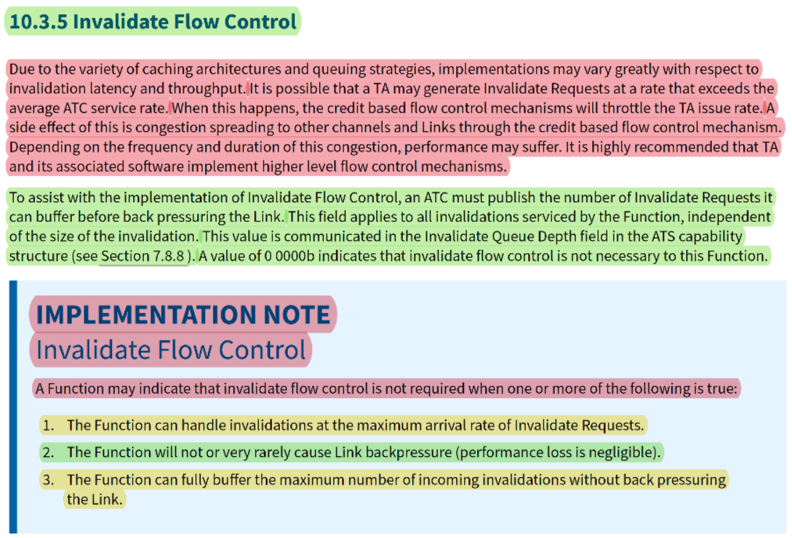
1. "A Function may indicate that invalidate flow control is not required when one or more of the following is true:"
2. "The Function can handle invalidations at the maximum arrival rate of Invalidate Requests."
3. "The Function will not or very rarely cause Link backpressure (performance loss is negligible)."
4. "The Function can fully buffer the maximum number of incoming invalidations without back pressuring the Link."

The sentences under "Requirements" specify conditions or actions that are necessary for implementing Invalidate Flow Control. The sentences under "Non-requirements" provide additional information but do not impose any specific requirements.

Figure 7. The GPT 3.5 model response

B. Model performance assessment on PCIe specification

The PCIe v.5 specification has been parsed into sentences in a similar way. All the obtained text fragments have been fed into the model in inference mode. Each fragment has been assigned with a probability value of being a requirement, i.e., the higher the value, the higher the chance that the text fragment is a requirement. Then all the text fragments in PDF file corresponding to the classified once have been highlighted in PDF as follows: green – if the probability is higher than 0.6, yellow – if the probability is between 0.2 and 0.6, red – if the probability is lower than 0.2. The example of this process is shown in Fig. 8.


10.3.5 Invalidate Flow Control

Due to the variety of caching architectures and queuing strategies, implementations may vary greatly with respect to invalidation latency and throughput. It is possible that a TA may generate Invalidate Requests at a rate that exceeds the average ATC service rate. When this happens, the credit based flow control mechanisms will throttle the TA issue rate. A side effect of this is congestion spreading to other channels and Links through the credit based flow control mechanism. Depending on the frequency and duration of this congestion, performance may suffer. It is highly recommended that TA and its associated software implement higher level flow control mechanisms.

To assist with the implementation of Invalidate Flow Control, an ATC must publish the number of Invalidate Requests it can buffer before back pressuring the Link. This field applies to all invalidations serviced by the Function, independent of the size of the invalidation. This value is communicated in the Invalidate Queue Depth field in the ATS capability structure (see Section 7.8.8). A value of 0 0000b indicates that invalidate flow control is not necessary to this Function.

IMPLEMENTATION NOTE

Invalidate Flow Control

A Function may indicate that invalidate flow control is not required when one or more of the following is true:

1. The Function can handle invalidations at the maximum arrival rate of Invalidate Requests.
2. The Function will not or very rarely cause Link backpressure (performance loss is negligible).
3. The Function can fully buffer the maximum number of incoming invalidations without back pressuring the Link.

Figure 8. The results of the model on PCIe specification PDF file.

The experiment on requirements identification for an experienced engineer has been also conducted. The estimation of the experienced verification engineer's performance is around 0.95 in both, precision and recall. Therefore, the human can be more successful in requirements recognition, but needs more time to decide whether the text is a requirement or not. The engineer identified requirements within five pages of highlighted and usual (without a model pre-processing) PDF file. The task on the highlighted text has been completed within 20% less time than on the usual one. This experiment has a very limited number of tests and a limited number of participant and cannot be considered robust. Furthermore, some of the requirements can be contradictory or related to not implemented parts of the system. The engineer handles these situations better as he/she can discuss it with colleagues for the better requirement management. Despite, the proposed approach saves time and provides valuable inputs for an engineer to provide a better decision making.

V. IMPLEMENTATION

The proposed model can be integrated into the automation system for technical documents analysis. The block diagram of the proposed automatic system is shown in Fig. 9.

The system takes a technical document as an input and can operate in two modes: training and inference. In the training mode, the document should be labeled, i.e., the text within the document should be appended with labels. In the simplest case, the labels can be 0 (the piece of text does not have a required characteristic) or 1 (the piece of text has a required characteristic) or a probability value (the value from 0 to 1) meaning the likelihood of the text having the required characteristic (the higher value means the higher likelihood). In the experiment conducted in this paper, required characteristic is non-requirement (0) / requirement (1). In the inference mode, the document has no labels in order to allow the system to make a prediction whether the parts of the text within the document have required characteristic or not. The system in the inference mode also outputs a probability value for each part of the document as a prediction.

The **Parser** block is used to analyze the content of the document (with or without labels) in order to convert the document into the list of text fragments (sentences, paragraphs, pages, etc.) with or without labels. Since documents often contain figures, the parser also extracts image data and converts it to the meaningful text data with or without labels and/or analyzes the contents of the image. The **Data Preparator** block connects text data with corresponding labels provided by the **Parser** block. If the system operates in inference mode, it just converts text data to the format expected by the Machine Learning Model (**ML Model**) block. The **ML Model** block loads the **Model Weights** from the internal system storage. In the training mode, the **ML Model** block executes the training process based on the dataset provided by the **Data Preparator** block. As a result, **Model Weights** should be updated within the optimization process in order to improve the quality of the **ML Model**, which is determined by the performance metric (accuracy, precision, recall, etc.). The updated weights are stored internally after the training process is over. In the inference mode, the **ML Model** block makes a prediction for each text fragment provided by the **Data Preparator** block. The result of the prediction can be a probability value (the value from 0 to 1) or a binary value (0 for the piece of text not having a required characteristic and 1 for the piece of text having a required characteristic). In the inference mode, the **ML Model** block passes the dataset (text fragments) with prediction results to the **Documents Processor** unit. This block converts source unlabeled document into a labeled document based on the source text data from the document and prediction results obtained from the **ML Model** block. The system can provide two types of outputs, i.e., in the training mode, it is an acknowledgement that the model weights are updated (1) or not (0) and in the inference mode, it is the labeled document, which is formed based on the initial document given as an input, and labels (prediction results) received from the **ML Model**.

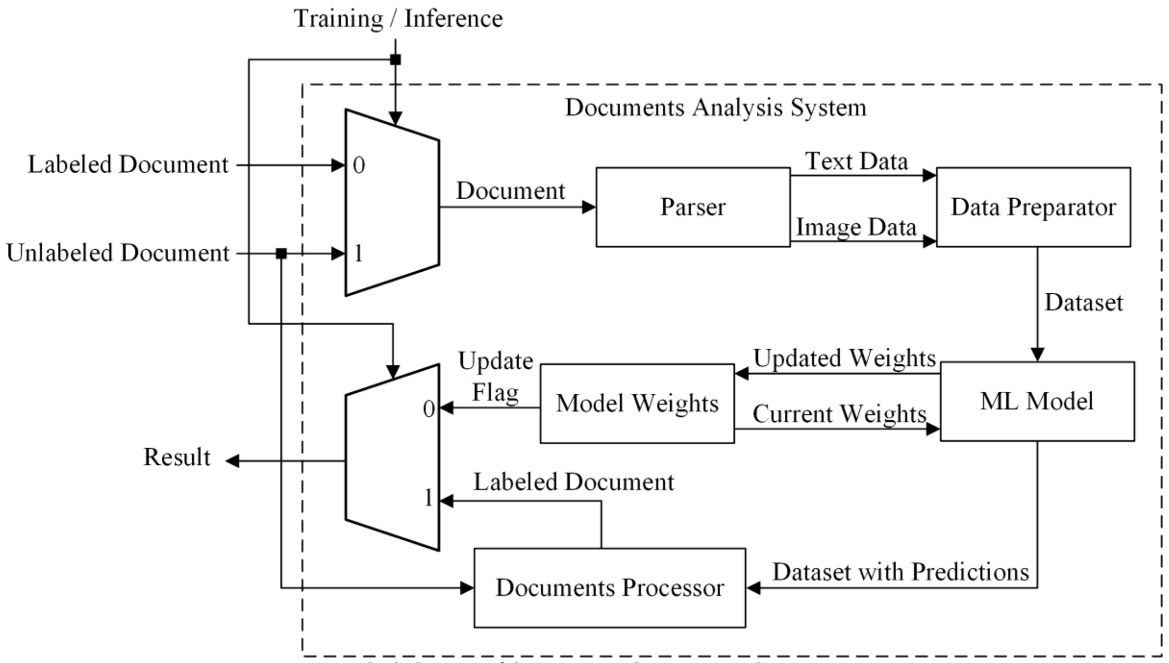


Figure 9. Block diagram of the automatic documents analysis system

The proposed system is implemented as a software using Python programming language. The implementation of the **Parser** and the **Documents Processor** blocks are based on PyPDF2 library [20]. The **Data Preparator** block converts extracted text data into the data frames (e.g., Pandas [21]). The **ML Model** is implemented using HuggingFace library [22]. The **Parser** block also includes an image-to-text convertor based on visual attention [23].

VI. CONCLUSION

The M-PHY 4.1 specification has been analyzed by a group of engineers in order to design a VIP. This data has been also found useful in order to automate requirements analysis process. The training dataset has been formed based on M-PHY specification analysis and utilized for training the LLM (GPT-2). The model has shown promising results (Precision=0.618 and Recall=0.874) that allow to improve the process of verification IP development, as requirements identification becomes easier compared to the manual way of doing it.

The proposed approach has been implemented as an automated documents analysis system, which can operate in training and inference modes. In the training mode, the system parses the contents of the documents and extracts labeled data for model training. In the inference mode, the system parses the document without labels and classifies its contents as requirements and non-requirements by providing a probability value for each text fragment.

So far, the model cannot fully automate the process of requirement identification, but it improves the performance of this process. Moreover, the proposed approach can be also used in order to improve other aspects of VIP development, such as suggesting tests, test groups, checkers or other verification means for particular requirements.

REFERENCES

[1] Srivastava, Amit, Rudra Mukherjee, Erich Marschner, Chuck Seeley, and Sorin Dobre. "Low Power SoC Verification: IP Reuse and Hierarchical Composition using UPF." DVCon proceedings (2012).

[2] Maropoulos, Paul G., and Dariusz Ceglarek. "Design verification and validation in product lifecycle." CIRP annals 59, no. 2 (2010): 740-759.

[3] PCI-SIG, "PCI Express 5.0 Specification.", available: https://pcisig.com/specifications (2023).

[4] MIPI Alliance "MIPI M-PHY Specification.", available: https://www.mipi.org/specifications/m-phy (2023).

[5] Hugging Face "OpenAI GPT2.", available: https://huggingface.co/docs/transformers/model_doc/gpt2 (2023).

[6] Ferrari, A., Spagnolo, G. O., & Gnesi, S. (2017, September). PURE: A dataset of public requirements documents. In 2017 IEEE 25th International Requirements Engineering Conference (RE) (pp. 502-505). IEEE.

[7] Ryan, Kevin. "The role of natural language in requirements engineering." [1993] Proceedings of the IEEE International Symposium on Requirements Engineering. IEEE, 1993.

[8] Sayyad Shirabad, J. and Menzies, T.J. (2005) The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada. Available: http://promise.site.uottawa.ca/SERepository

[9] Hull, Elizabeth, et al. "DOORS: a tool to manage requirements." Requirements engineering (2002): 187-204.

[10] Dias Canedo, Edna, and Bruno Cordeiro Mendes. "Software requirements classification using machine learning algorithms." Entropy 22.9 (2020): 1057.

[11] Khayashi, Fatemeh, et al. "Deep Learning Methods for Software Requirement Classification: A Performance Study on the PURE dataset." arXiv preprint arXiv:2211.05286 (2022).

[12] Ivanov, Vladimir, et al. "Extracting Software Requirements from Unstructured Documents." International Conference on Analysis of Images, Social Networks and Texts. Cham: Springer International Publishing, 2021.

[13] Luttmer, Janosch, et al. "Requirements extraction from engineering standards–systematic evaluation of extraction techniques." Procedia CIRP 119 (2023): 794-799.

[14] Chang, Yupeng, et al. "A survey on evaluation of large language models." arXiv preprint arXiv:2307.03109 (2023).

[15] DeepLearning.AI "A Complete Guide to Natural Language Processing.", available: https://www.deeplearning.ai/resources/natural-language-processing/ (2023).

[16] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[17] Brown, Tom, et al. "Language models are few-shot learners." Advances in neural information processing systems 33 (2020): 1877-1901.

[18] Touvron, Hugo, et al. "Llama: Open and efficient foundation language models." arXiv preprint arXiv:2302.13971 (2023).

[19] OpenAI Inc. "Models. GPT 3.5", available: https://platform.openai.com/docs/models/gpt-base (2023).

[20] Python Software Foundation "PyPDF2 3.0.1", available: https://pypi.org/project/PyPDF2/ (2023).

[21] The pandas development team. Pandas-dev/pandas: Pandas (v2.1.0). Zenodo, available: https://doi.org/10.5281/zenodo.8301632 (2023).

[22] Wolf, Thomas, et al. "Huggingface's transformers: State-of-the-art natural language processing." arXiv preprint arXiv:1910.03771 (2019).

[23] Liu, X., and M. Milanova. "Visual attention in deep learning: a review." Int Rob Auto J 4.3 (2018): 154-155.