Extracting Domain Models from Textual Requirements in the Era of Large Language Models

Sathurshan Arulmohan
McMaster University
CAS & McSCert
Hamilton, Canada
arulmohs@mcmaster.ca

Marie-Jean Meurs
Université du Québec à Montréal
CIRST
Montreal, Canada
meurs.marie-jean@uqam.ca

Sébastien Mosser
McMaster University
CAS & McSCert
Hamilton, Canada
mossers@mcmaster.ca

***Abstract*—Requirements Engineering is a critical part of the software lifecycle, describing what a given piece of software will do (functional) and how it will do it (non-functional). Requirements documents are often textual, and it is up to software engineers to extract the relevant domain models from the text, which is an error-prone and time-consuming task. Considering the recent attention gained by *Large Language Models* (LLMs), we explored how they could support this task. This paper investigates how such models can be used to extract domain models from agile product backlogs and compare them to *(i)* a state-of-practice tool as well as *(ii)* a dedicated *Natural Language Processing* (NLP) approach, on top of a reference dataset of 22 products and $1,679$ user stories. Based on these results, this paper is a first step towards using LLMs and/or tailored NLP to support automated requirements engineering thanks to model extraction using artificial intelligence.**

***Index Terms*—Domain Modeling, Natural Language Processing, Large Language Models, Concept Extraction, User stories**

I. Introduction

Requirements Engineering (RE) is essential to the software lifecycle. RE is a spectrum between two extremes: included as an early step in the waterfall/V-cycle models or advocated as an on-the-fly activity by the Agile community (*e.g.*, with user stories backlogs). In both cases, requirements artifacts are produced using natural language to express the requirements to which the software under construction must conform.

It is up to the requirement engineers to adequately capture such requirements from stakeholders and "formalize" them (*e.g.*, with use case models in UML and associated scenarios, formal specifications, or a product backlog according to agile methods). Then, software designers have to translate these artifacts (which mainly consist of natural language) into actionable design artifacts. This step is tricky, as ambiguities/conflicts can exist despite all efforts put into the RE step.

This is where modelling and *Natural Language Processing* (NLP) come to help [1]: automated techniques can help extract models from these natural language artifacts [2]. By removing "noise" and increasing "signal", software engineers can focus on the abstracted models to identify ambiguities, conflicts and validate the completeness of requirements concerning the involved stakeholders. Consequently, the automated support of model extraction from requirements artifacts is essential to provide good support for engineers.

This work is funded by the *Natural Sciences and Engineering Research Council of Canada* (NSERC) under the *Discovery Grant* (DG) program and McMaster's Faculty of Engineering (*Excellence in Research Award*)

Recently, *Large Language Models* (LLMs) such as GPT [3] and conversational agents such as ChatGPT [4] started to claim to be breakthrough enablers for various tasks, including software development. **In this paper, we propose to investigate how LLMs and conversational agents can support model extraction from requirements, focusing on agile backlogs** as these are, in essence, only based on textual artifacts written in natural language. We leverage here our expertise gained in previous work on model composition for requirement artifacts [5]. We start by giving a comprehensive background on such backlogs and defining a baseline for evaluation using a reference dataset for backlogs and an extraction tool from the state of practice in Sec. II. We then describe how GPT-3.5 can be used in conversational mode to support such a task Sec. III, and Sec. IV does the same using a tailored approach developed with NLP experts. In Sec. V, we evaluate how the three approaches compare to each other, and Sec. VI concludes this paper.

II. Background: Agile-driven Requirements

This paper focuses on *agile* backlogs to support requirements definition. As motivated in the previous section, this choice is driven by multiple factors. First, being a mainstream approach to express requirements in open-source software, it gives us access to reference requirements, like the one compiled by Dalpiaz *et al.* [6] in 2018. Despite this paper focusing on agile backlogs, it would be easy to transfer the experimental contribution to *classical* requirements documents and compare the result with other tooling, depending on the availability of such data. We emphasize that our focus here is to support the accessibility of the datasets and tools (see Sec. V) to support reproducibility, and, as a consequence, we ruled out of scope any dataset or tooling that would rely on proprietary/confidential information.

A. Related Work

A survey was published in 2021 by Zhao *et al.* [7] to answer several questions, such as "*What is the focus of NLP4RE research?*" (RQ3) and "*What is the state of tool development in NLP4RE research?*" (RQ4). 64.59% of the studies were working on requirement specification as input. Among 370

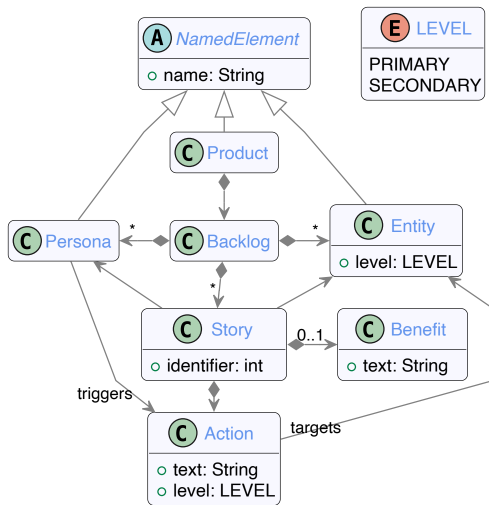


Fig. 1. Metamodel for Agile Backlogs Domains

studies, they identified 59 studies related to modelling and 63 related to extraction (RQ4). From a tooling perspective, they have identified 130 tools used in their collected studies. Among this set, modelling tools represent 34 tools, and extraction tools 24, representing 44% of the available tooling. In total, 140 different NLP techniques are used in the corpus of tools identified in this survey.

As the survey was published in a pre-GPT world, none of the studies considered GPT per se, and only a few used LLMs such as BERT. Since LLMs became popular, we can see some preliminary results published as pre-prints or conference papers. This paper mainly focuses (so far) on recommendations more than extraction, such as the work by Weyssow *et al.* [8] used to recommend concepts during modelling. Related to extraction, Bajaj *et al.* [9] concludes that GPT-3 outperforms classical tools from the state of practice concerning use case extraction. These preliminary results seem promising, so investigating how LLMs can be used to support extraction seems legit.

B. Processing User stories Backlogs

According to agile methods, requirements are written as a prioritized set of *user stories*, organized in a *product backlog*. A *story* represents, using natural language, a tuple containing the following information: *(i)* the *persona* involved in the story, *(ii)* the *actions* this persona will perform on the system, *(iii)* the *entities* involved in the actions, and, optionally, *(iv)* a *benefit* obtained by the persona after having completed these actions. Classically, user stories are expressed using the *Connextra Template* [10], under the form: "*As a <**PERSONA**>, I can <**ACTIONS** over **ENTITIES**>, so that <**BENEFIT**>*". Even if other templates exist, stories following template-based writing simplify the concept extraction tasks. In FIG. 1, we depict the associated metamodel to represent a product backlog. It is important to note that this paper focuses on concept extraction according to this metamodel.

To support the product development phase, extracted concepts are classically organized into analysis and design models (*e.g.*, class diagrams, sequence diagrams) [11]. We consider these post-extraction transformations out of scope and focus our contribution on the extraction task. By extracting personas, actions and entities, tools can transform the extracted material into their chosen domain model representation using ontologies or domain-oriented class diagrams.

C. Annotating User Stories as a Ground Truth

Considering our choice of only using publicly available requirements, we focused on the dataset published by Dalpiaz *et al.*. It contains 22 product backlogs and 1,679 user stories that the requirements engineering community curated. The dataset is published as a raw archive, *i.e.*, an archive of 22 text files, each containing the user stories associated with the product it describes, one per line. To the best of our knowledge, no publicly available expert-based annotation establishes a ground truth for the different concepts described in the dataset.

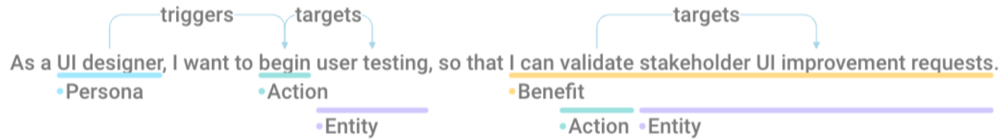
As a consequence, we first started by manually annotating the dataset, using the Doccano[1] platform, as a *Named Entity Recognition* task. We loaded the different labels (*i.e.*, *Persona*, *Action*, *Entity*, *Benefit*) and two relations (*i.e.*, *triggers*, *targets*) used in out domain metamodel, as depicted in FIG. 2.

The annotation process was the responsibility of the 1st author. To ensure the quality of the annotated set, we introduced several cross-checks: *(i)* an initial calibration was done in collaboration with the 3rd author on 75 stories randomly sampled, *(ii)* validation meetings were held on a bi-weekly basis over two months during the annotation phase, and *(iii)* the 3rd author cross-checked manually 330 randomly sampled annotated stories (19.6%). No significant deviations were found during this step, as only 21 stories containing (minor) *disputable* elements were identified, leading to an inter-rater agreement of 94% for this sample, which is classically considered as excellent. In case of significant disputes, the 2nd author would have acted as an external referee and, as such, was kept out of the annotation process (SEC. V-C). Overall, the annotation phase lengthed two months, and consumed in total 10 days for the 1st author, and 1.5 days for the 3rd author (not counting the bi-weekly meetings). The produced ground truth is available as part of our open-data repository [12].

D. Creating a Baseline from the State of Practice

We complemented the ground truth by running *Visual Narrator* (VN [11]) on top of the dataset and obtaining a baseline from the state of practice. VN is an open-source software designed to explicitly extract *conceptual models* from user stories backlog. For each story, it extracts (using an ontological approach) the same elements as the ones described in our domain metamodel. We ran VN on top of the provided backlogs and extracted the concepts it extracts by parsing the output files it produces. Internally, VN relies on *spaCy* [13] for natural language processing, using the `en_core_web_md` model for English. The tool also leverages the Connextra

[1] https://doccano.herokuapp.com/



This story is extracted from backlog #02. In this backlog, "user testing" is a first-class entity understood as a synonym for "test reports summary".

Fig. 2. Example of annotated user story using Doccano Annotation UI

template by looking for special keywords[2] (*e.g.*, "As a") to guide its extraction process.

After this step, we obtain a new annotated dataset containing the results of a domain model extraction on top of the input corpus.

III. A Conversation with (Chat)GPT 3.5

We experimented with ChatGPT to find the best way to extract the information from a user story backlog in a *"rapid prototyping"* way. When we obtained satisfying results through the Web interface, we translated the prompts we were using into calls to the API and used a backlog of 25 stories to run the extraction and manually check the results. The following section describes in a step-by-step way how we ended up prompting GPT the way we did for the experiment, as we defend it is part of the contribution of this paper to describe how LLMs can be interacted with to obtain results.

A. Prompt Engineering

Initially, we designed the extraction as a *batch* approach: we asked the GPT agent to extract concepts, categories and relations from a set of stories. After explaining the extraction task to the agent, we provided a complete backlog with the hope that this comprehensive vision of the backlog would support good results. Unfortunately, this faced two immediate limitations. First, a query (prompt plus response) to the GPT API is limited to $4,097$ tokens, representing an average of $3,000$ words. This limit is not a problem for ongoing product backlogs (which are classically maintained below 100 stories[3] by product owners) but would become one if someone is interested in extracting information from legacy software. The second issue was identified during the manual quality check. With this approach, the agent was fabricating data (*i.e.*, the so-called "hallucinations"), for example, by mixing elements between stories. The response would claim that story S_i refers to Persona P_j, even if the story does not mention P_j. In some cases, P_j was not even mentioned anywhere in the backlog.

To limit this, we entered a second iteration of prompt engineering. Both problems seemed triggered because we provided the agent with a complete backlog. Consequently, we transformed our prompt approach for the following: provide stories one by one, ask GPT to extract concepts, categorize them, and extract relations for each story independently. This approach drastically limited "hallucinations", as the answer mainly referred to elements in the provided story. It also fixed the request length issue. Unfortunately, it does not produce usable results. GPT produced texts that did not respect the input constraints (we tried JSON or CSV) and mixed up things, *e.g.*, categorizing an entity as a persona. When not constrained, each request led to a different output format (*e.g.*, the CSV row was named `Entities`, `Entity`, and positioned at different places). When constrained, the system was not following the instructions, mixing up rows and concepts.

To make this more straightforward and usable, we started investigating the *function calls* mechanisms introduced in `gpt-3.5-turbo-0613`. This version of GPT 3.5 was released on June 13, 2023. It became the *standard version* for GPT 3.5 on June 27, 2023. Function calls are designed in the API to give control to the output produced by the system. Developers can provide a JSON schema to model their response, and the system will use this schema and *"fill in the blanks"* instead of regular text generation. For example, if one expects their answer to be an array of strings containing the name of the personas, they can provide a schema inside their request, and GPT will use it as output format (as in Lst. 1, lines $3 \rightarrow 12$). An example of such a constrained response is described in Lst. 2. Instead of providing a `content`, the agent answers with a `function_call`, providing its name and how it should be called (`arguments`). When the engine uses a function call as an answer, the conversation is immediately ended (`"finish_reason": "function_call"`). Consequently, obtaining a sequence of calls using this technique is impossible, as the engine will stop once a call is returned in the answer.

Given this limitation, we faced two options: *(i)* defining a large schema containing all the information we wanted to extract from a user story, or *(ii)* engaging in a conversation with the model. Without surprise, the first approach was inconclusive, as GPT was mixing up elements in the way it was filling in the blanks of the schema (*e.g.*, using personas as entities or, more surprisingly, as actions). As this API is designed conversationally, we opted for the second approach. We finally designed the processing of each story as a conversation with the model, with each answer constrained by a dedicated (and smaller) schema [14].

B. Extracting domain models with GPT 3.5

Building on the different approaches described in the previous section, we used the following protocol to extract concepts from user stories. We handle each story independently, and for each story, we engage in a conversation with GPT 3.5 and

[2] https://github.com/MarcelRobeer/VisualNarrator/blob/master/lang/en/indicators.py

[3] https://resources.scrumalliance.org/Article/scrum-anti-patterns-large-product-backlog

```
response = openai.ChatCompletion.create(
    model = "gpt-3.5-turbo-0613",
    functions = [ # constraining GPT with a schema
      { "name": "record_elements", # Function to be called in the response
        "description": "Record the elements extracted from a story",
        "parameters": { # Signature description
          "type": "object",
          "properties": {
            "personas": {
              "type": "array",
              "description": "The list of personas extracted from the story",
              "items": { "type": "string" }}}}],
    messages = conv,
    temperature=0.0) # To make the answer deterministic (as much as possible)
```

Listing 1. Calling GPT 3.5 and specifying a function call argument

```
{
  ...
  "choices": [{
    "index": 0, "message": {
    "role": "assistant", "content": null,
    "function_call": {
      "name": "record_elements",
      "arguments":
        "{\"personas\": [\"repository manager\"]}"
    }},
  "finish_reason": "function_call"}],
  ...
}
```

Listing 2. Example of answer from the API (execution of LST. 1)

constrain its responses using the function call mechanism. We have organized the conversation into four phases:

1) **Setup**. First, we impersonate the *system* role and ask the engine to adopt a persona. We describe the global task that will be asked later to set up the request execution context.
2) **Concepts**. Still using the *system* role, we now specify with more detail the task to extract personas, entities, actions and benefits from a story. We then provide an example of such an extraction using one of our manually annotated stories. Finally, switching to *user* role, we provide the <STORY> to process.
3) **Categorization**. As the model is stateless, we have to inject the answer from the previous phase into the conversation. Thus, we impersonate the *assistant* role and add a conversation entry describing the <CONCEPTS> previously obtained. Following the same pattern as in the previous phase, we describe the task using the *system* role (categorizing primary and secondary actions and entities) and provide an example of such an execution.
4) **Relations**. The final step uses the same pattern. We first inject the <CATEGORIES> as the *assistant*, and then describe the task and provide an example as the *system*.

As of June 2023, OpenAI lists as best practices six strategies and 17 tactics[4] to support prompt engineering. The conversation we used is described in TAB. I and follows the relevant OpenAI tactics. We implemented (see TAB. II) completely seven of the provided guidelines (✓), three partially (∼), and the remaining seven were not relevant to our problem (NA). The complete code is available on GitHub [14].

[4]https://platform.openai.com/docs/guides/gpt-best-practices

C. Discussion & Lessons learned

In this section, we only discuss the *technical* dimensions of using the GPT 3.5 API to support the extraction task. We will discuss the quality of results in SEC. V.

- **Ease of use**. Based on the available documentation and documented best practices, interacting with GPT 3.5 is straightforward and does not require specialized programming skills: basic Python programming skills are sufficient. Using ChatGPT to support fast prototyping is also very helpful, as it allows one to quickly interact with the LLM in a trial/error way.
- **Privacy**. Being a hosted LLM, each story processed by GPT 3.5 is sent to OpenAI. For a publicly available dataset, it might not be considered an issue, but this should be considered for backlogs containing proprietary information (usually the case for commercial products). OpenAI enacted (05/23) a policy stating that they are no longer using users' data for upcoming training/improvements of their model. This policy might not be sufficient for some sovereign data, for example. This can be mitigated by deploying an on-premise LLM.
- **"Hallucination"**. Being a *stochastic parrot* [15] by design, GPT does not understand the concepts it extracts. Thus, even if careful prompt engineering can limit this phenomenon, it commonly misuses concepts, *e.g.*, consider an element a persona and then an entity, despite these two being exclusive concepts.
- **JSON Schema**. Constraining the answer to fit a given JSON schema helps support the automation of extracted data. Out of $5,193$ calls to the model required to process our backlog dataset, only two responses (0.03%) were invalid JSON. An inconvenient of the approach is that it complexifies the interactions with the model and requires more programming skills. We also encountered some situations where the generated JSON document was missing elements (*e.g.*, in a relation, the kind of relation –trigger or target– was not specified).
- **Cost**. ChatGPT, used for prototyping, is free to use. When calling the API, you need a valid account, and the system invoices users on a *pay-as-you-go*. As of June 23, the chat completion API (`gpt-3.5-turbo`) is offered for $\$0.0015/1k$ input tokens. Processing the whole dataset consumed $2,228,162$ input tokens (85%) and produced $387,324$ completions tokens as output (15%). The overall

TABLE I
PROMPTS USED TO EXTRACT CONCEPTS FROM USER STORIES USING GPT-3.5

Phase	Role	Prompt
Setup	System	You are a requirements engineering assistant specialized in agile methods and backlog management.
	System	You will be provided by the user a user story, and your task is to extract elements from these models and call provided functions to record your findings.
	System	You are only allowed to call the provided function in your answer.
Concepts	System	The elements you are asked to extract from the stories are the following: Persona, Action, Entity, and Benefit. A Story can contain multiple elements in each category.
	System	Here is an example. In the story '*As a UI designer, I want to begin user testing, so that I can validate stakeholder UI improvement requests*', the Persona is '*UI designer*'. The actions are '*begin*' and '*validate*'. The entities are '*user testing*' and '*stakeholder UI improvement requests*'. The benefit is '*I can validate stakeholder UI improvement requests*'.
	User	Here is the story you have to process: **<STORY>**
Categorization	Assistant	Here are the extracted concepts: **<CONCEPTS>**
	System	You now need to make the difference between primary concepts and secondary concepts in the information you have extracted.
	System	In the example that was given initially, the actions primary action is 'begin' and the secondary one is 'validate'. The primary entity is 'user testing' and the secondary entity is 'stakeholder UI improvement requests'.
Relations	Assistant	Here is the categorization: **<CATEGORIES>**
	System	You now need to extract relationships betwen personas and actions (named trigger), and between actions and entities (named target).
	System	In the example that was given initially, the persona 'UI designer' triggers the action 'begin', and the action 'begin' targets the entity 'user testing'.

TABLE II
OPENAI'S TACTICS TO SUPPORT PROMPT ENGINEERING (JULY 2023)

Strategy #1: Write clear instructions		
T_1	Include details in your query to get more relevant answers	✓
T_2	Ask the model to adopt a persona	✓
T_3	Use delimiters to clearly indicate distinct parts of the input	✓
T_4	Specify the steps required to complete a task	✓
T_5	Provide examples	✓
T_6	Specify the desired length of the output	NA
Strategy #2: Provide reference text		
T_7	Instruct the model to answer using a reference text	$\sim$
T_8	Instruct the model to answer with citations from a reference text	NA
Strategy #3: Split complex tasks into simpler subtasks		
T_9	Use intent classification to identify the most relevant instructions for a user query	✓
T_{10}	For dialogue applications that require very long conversations, summarize or filter previous dialogue	✓
T_{11}	Summarize long documents piecewise and construct a full summary recursively	NA
Strategy #4: Give GPTs time to "think"		
T_{12}	Instruct the model to work out its own solution before rushing to a conclusion	NA
T_{13}	Use inner monologue or a sequence of queries to hide the model's reasoning process	$\sim$
T_{14}	Ask the model if it missed anything on previous passes	NA
Strategy #5: Use external tools		
T_{15}	Use embeddings-based search to implement efficient knowledge retrieval	NA
T_{16}	Use code execution to perform more accurate calculations or call external APIs	NA
Strategy #6: Test changes systematically		
T_{17}	Evaluate model outputs with reference to gold-standard answers	$\sim$

cost to analyze the $1,679$ stories in the corpus is $\$4.12$ (USD), which can be considered neglectable.

- **Reliability**. Interacting with the model at scale required more engineering than the documentation described. While running the experiment, we encountered server-side errors (*e.g.*, BadGateway – HTTP 502, ServiceUnavailableError – HTTP 500) with a 2.78% rate: out of $5,193$ calls, 144 were in error. To fix this, we introduced a *Circuit Breaker*[5] in the code. Introducing such a pattern makes the code more complex and requires some solid programming skills.
- **Response time**. When processing the first backlogs, we encountered a response time of up to 45 minutes for a single request. We fixed this by manually introducing a client-side timeout mechanism, voluntarily interrupting a call if it took more than 30 seconds. It drastically improved the time required to process the upcoming backlogs to the cost of a more complex code. Processing the complete dataset consumed 200 minutes and six seconds, equivalent to almost three hours and a half, with no training required.

IV. USING A DEDICATED NLP APPROACH (CRF)

Before jumping to conclusions, we decided to include in the comparison of the results an upper limit, being a dedicated *Natural Language Processing* (NLP) approach, by teaming up with an NLP group (represented by the 2nd author).

Where VN combines rule-based extraction and *spaCy* language model, it misses the point that extracting domain models from user story backlog can be modelled as a contextualized pattern recognition task. This representation allows us to use approaches known for being more efficient and accurate. The

[5] https://martinfowler.com/bliki/CircuitBreaker.html

fact that we built a ground truth of $1,679$ user stories also allows us to rely on supervised learning approaches.

A. Introducing Conditional Random Fields (CRF)

CRFs [16] are a particular class of *Markov Random Field*s, a statistical modelling approach supporting the definition of discriminative models. They are classically used in pattern recognition tasks (labelling or parsing) when context is important to identify such patterns.

To apply CRF to our task, we need to transform a given story into a sequence of tuples. Each tuple contains minimally three elements: *(i)* the original word from the story, *(ii)* its syntactical role in the story, and finally *(iii)* its semantical role in the story. The syntactical role in the sentence is classically known as *Part-of-Speech* (POS), describing the grammatical role of the word in the sentence. The semantical role plays a dual role here. For training the model, the tags will be extracted from the annotated dataset and used as target. When used as a predictor after training, these are the data we will ask the model to infer. Consider the following example:

$$S = [\text{'As'}, \text{'a'}, \text{'UI'}, \text{'designer'}, \text{','}, \ldots] \quad (1)$$

$$POS(S) = [\text{ADP}, \text{DET}, \text{NOUN}, \text{NOUN}, \text{PUNCT}, \ldots] \quad (2)$$

$$Label(S) = [\emptyset, \emptyset, \text{PERSONA}, \text{PERSONA}, \emptyset, \ldots] \quad (3)$$

S represents a given user story (FIG. 2). $POS(S)$ represent the Part-of-speech analysis of S (here using *spaCy*, the same library used by Visual Narrator). The story starts with an adposition (ADP), followed by a determiner (DET), followed by a noun, followed by another noun, $\ldots$. Then, $Label(S)$ represents what we are interested in: the first two words are not interesting, but the 3rd and 4th words represent a *Persona*. A complete version of the example is provided in FIG 3.

The main limitations of CRF are that *(i)* it works at the word level (model elements can spread across several words), and *(ii)* it is not designed to identify relations between entities.

To address the first limitation, we use a glueing heuristic. Words that are consecutively associated with the same label are considered as being the same model element, *e.g.*, the subsequence ['UI', 'designer'] from the previous example is considered as one single model element of type `Persona`. We apply this heuristic to everything but verbs, as classically, two verbs following each other represent different actions. Again, we use a heuristic approach to address the second limitation. We bound every `Persona` to every primary `Action` (as `trigger` relations), and every primary `Actions` to every primary `Entity` (as `target` relations).

B. Implementing the task using CRF

As VN is implemented in Python 3.7 and relies on *spaCy* for POS tagging, we decided to use the same stack to reduce the comparison gap. Among the different implementations of CRF available, `sklearn-crfsuite`[6] was the most suitable for this reason. Unfortunately, the code was outdated (2014) and not maintained (referring to version 0.15 of scikit-learn). Thus, we fixed the integration problems so that the code could run in the recent scikit-learn version (1.x) and with a recent version of Python (3.10). This version of the scikit-learn plugin is available as an open source software [17]. The CRF tool is available on GitHub[7]

Training the model is straightforward as soon as the CRF framework is integrated. We transform each story into a stream of words, compute the POS using *spaCy*, associate the semantic label from the annotated dataset, and use this to create our feature set used for training. The feature set represents the context windows by associating a given word with its POS, label, and information on the surrounding words. In addition to the POS and labels described in the previous section, we also provide the algorithm with the length of the word, its capitalization, and its alphanumeric status. Our current implementation has a sliding window of four words preceding the current one and one following it. These values were selected with an experimental approach to minimize overfitting while maintaining a reasonable F-measure (see SEC. V). When the model is trained, we can now provide an unknown story to the model and obtain a sequence of semantic labels as output. Using the previously described heuristics, we transform these labels into instances of our domain metamodel to obtain the final result.

C. Discussions & Lessons Learned

As for GPT, we only discuss the lessons learned at the technical level.

- **Domain Knowledge**. We could only implement this approach thanks to a close collaboration with NLP experts. The landscape of NLP approaches is wide, and without domain expertise, we would have never identified CRF as a potential candidate for domain model extraction. Being able to transfer our software engineering problems to AI experts is essential to avoid reinventing a squared wheel by simply aggregating *visible* solutions.
- **Fighting the hype**. As stated, CRF is losing popularity. The hype/fashion lifecycle is classical in software engineering. Think about Aspect-oriented Programming: supposed to revolutionize software development in the '00s, they are now forgotten, except in particular places where they perfectly fit (*e.g.*, Spring, a reference framework to develop enterprise code in java, intensively rely on aspects under the hood). It is important to build on the shoulders of giants and not to succumb to the latest fashionable "thing".
- **Technical Complexity**. Compared to the simplicity of the GPT code, implementing this CRF approach required more advanced Python programming skills, including fixing outdated dependencies, patching a sci-kit-learn module, and integrating the approach in a machine learning pipeline. This is not unachievable, but the effort and skills required are clearly different by an order of magnitude.
- **Cost**. Being self-hosted, the approach does not directly cost money. However, the training time and resource

[6]https://github.com/TeamHG-Memex/sklearn-crfsuite

[7]https://github.com/ace-design/nlp-stories/tree/main/nlp/nlp_tools/crf

Word	As	a	UI	designer	,	I	want	to	begin	user	testing	,
POS	ADP	DET	NOUN	NOUN	PUNCT	PRON	VERB	PART	VERB	NOUN	NOUN	PUNCT
Label	–	–	PER	PER	–	–	–	–	P-ACT	P-ENT	P-ENT	–

Word	so	that	I	can	validate	stakeholder	UI	improvement	requests	.
POS	SCONJ	SCONJ	PRON	AUX	VERB	NOUN	NOUN	NOUN	NOUN	PUNCT
Label	–	–	–	–	S-ACT	S-ENT	S-ENT	S-ENT	S-ENT	–

POS tags are the Universal POS tags (https://universaldependencies.org/u/pos/), computed by spaCy.
Labels: PER (Persona), P-ACT (Primary Action), P-ENT (Primary Entity), S-ACT (Secondary Action), S-ENT (Secondary Entity)

Fig. 3. Minimal Feature Set, associating part-of-speech (POS) and semantic labels to each word in a given story

consumption need to be considered. On an average laptop, the training takes less than a minute for a set of 1k stories.

V. Validation

This section empirically evaluates the three approaches (Visual Narrator, GPT-3.5 and CRF) to the ground truth established during the annotation phase.

A. Experimental Setup

First, we had to build the corpus that would be used to support the evaluation. We started by cleaning up the dataset from duplicated stories. Then, we consider the *intersection set* of the backlogs as the set of stories where all the tools considered were able to produce parsable outputs. Ultimately, we perform our evaluation with a corpus containing $1,459$ stories (87% of the initial corpus).

As CRF requires training, we used a classical 80/20 separation. We randomly split the dataset into a training set containing 80% of the stories and used the remaining 20% for evaluation with the three tools. We trained CRF in two modes: *(*i) global and *(ii)* individual. For the global mode, we extracted the training set from the complete corpus, mixing different products. The individual mode applied the 80/20 partition backlog by backlog (Fig. 4).

To properly compare results, we must define whether a result is good or bad. At best, a result is perfect when the AI approach produces precisely the same elements as the one available in the ground truth. We call this comparison mode the *Strict* one. To relax the constraints, we also considered a comparison where the AI tool produces a superset of the ground truth, checking that the baseline is *Included* in the produced result. Finally, we also considered a *Relaxed* comparison, where we relaxed the checking by considering plurals and singular equivalent to each other, or ignoring adjective qualifiers. Eventually, the results are consistent for each tool, whatever comparison measure is used. To avoid unfair comparisons, we do not compare the identified `trigger` and `target` relations, as VN was not designed for this task.

We represent in Fig. 4 the averaged F-measure (F_1) for each algorithm, comparison mode, and extracted model element. Being defined as the harmonic mean of precision and recall, F_1 considers correctly and incorrectly classified observations (as opposed to accuracy, which focuses on positive cases). It is classical to use F_1 instead of accuracy when the cost of making a wrong decision is essential (here, extracting domain concepts that are not part of the domain). An F_1 score of 1.0 means a perfect match, and a F_1 score of 0.0 means that precision or recall is null.

B. Result Interpretation

First, it is interesting to notice that VN (establishing a baseline from the state of practice) performs reasonably on `Personas` and `Actions` but quite poorly on `Entity`(ies), with a maximum $F_1 < 0.25$ (relaxed comparison).

Identifying the `Persona` instances is by far the most straightforward task. The Conextra Template can explain this, as their position in the user story is entirely predictable (adding noise in front of the "As a" marker in a story disturbs VN). Same for identifying the `Actions`, as their position is also predictable (*e.g.*, "I want to…"). An `Entity`, being a common *noun* that can appear anywhere in the sentence, is harder to detect by a rule-based system. VN often fails at identifying proper entities, and default to one named *System* (even if not mentionned in the story).

When comparing GPT-3.5 with VN, it is clear that using an LLM to support the concept extraction task creates a breakthrough. VN's implementation consists of a 23 Python file and $2,856$ lines of code, while interfacing with GPT-3.5 requires one single script of 337 lines. Furthermore, the obtained results outperform VN in every case. The most significant improvement relates to identifying `Actions`: VN F_1 score is around 0.2, whereas GPT-3.5 is around 0.6, without any need for training or annotation. Based on the results, LLMs are interesting for an average extraction, as they do not require additional effort to outperform the state of practice.

However, it is essential to note that GPT-3.5 was systematically outperformed by our CRF implementation (its worst F_1 value is 0.81, and it reaches a perfect 1.0 for `Personas`' extraction). This is no surprise: LLMs are a one-size-fits-all approach, where a tailored approach developed with a domain expert (here in NLP) to use the correct underlying model/algorithm is precisely addressing the problem to solve.

C. Limitations & Threats to Validity

First, the creation of the ground truth is, by nature, influenced by subjective factors. We mitigated this according to a precise protocol, including bi-weekly meetings and manual cross-check of 19.6% of the annotated dataset with an inter-rater agreement of 94%. However, the two authors involved in the annotation process might have introduced unconscious bias in their annotation guidelines. An external validation of the annotated dataset would be beneficial. This threat is mitigated by the fact that *(i)* the results are provided relatively to this ground truth, and *(ii)* the 3rd author has 11 years of experience teaching agile methods and user stories writing as an academic instructor and industrial consultant.

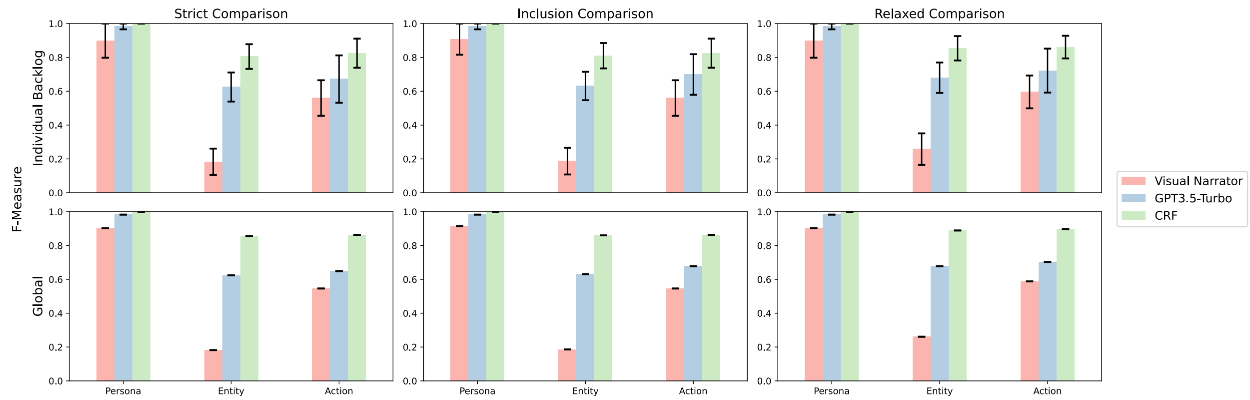


Fig. 4. Comparing approaches to the ground truth: F-measure results for Visual Narrator, GPT-3.5 and CRF

The second threat to validity is related to the LLM model we used and the prompt engineering method we followed. We only used GPT-3.5, and more extensive experimentations are needed to see if the results presented in this paper hold when compared to GPT-4 (OpenAI), Bard (Google), or LLaMA (Meta). Concerning prompt engineering, despite intensively relying on OpenAI best practices, as LLMs are black boxes, it is very hard to identify which elements influence the model positively or negatively. To support reproducibility, we also had to set GPT-3.5's temperature to 0, making it as deterministic as possible. This choice might have introduced a negative bias in the result, and more experiments are needed to handle the non-deterministic nature of LLMs to evaluate their results in the context of domain model extraction.

VI. Conclusions

In this paper, we have experimented with how Visual Narrator, GPT-3.5 and a CRF-based approach performed to automate the extraction of domain concepts from agile product backlogs. The first contribution of this paper is an open-data repository [12] containing an annotated version of a reference backlog corpus (22 products, $1,679$ user stories), reusable by other researchers. Then, for both GPT-3.5 and CRF, we provided an in-depth discussion of the engineering dimensions associated with their development (including prompt engineering for GPT-3.5), and we evaluated the results at scale on top of the annotated corpus.

Out of these results, if the GPT-3.5 LLM is better ($F_1 \approx 0.6$) than the state of practice concerning this task, it is outperformed by the CRF approach designed with NLP experts ($F_1 \approx 0.85$), which require less than a minute of training. It triggers a question for the following research efforts in this direction: *Should we, as researchers, choose to use easily integrable but far-from-ideal (energy cost, average F_1) solutions available off-the-shelf, or should we instead choose to invest in collaborating with AI/NLP domain experts to define tailored solutions that would support MDE intelligence?*

References

[1] F. Dalpiaz, A. Ferrari, X. Franch, and C. Palomares, "Natural language processing for requirements engineering: The best is yet to come," *IEEE Software*, vol. 35, no. 5, pp. 115–119, 2018.

[2] G. Mussbacher, B. Combemale, J. Kienzle, S. Abrahão, H. Ali, N. Bencomo, M. Búr, L. Burgueño, G. Engels, P. Jeanjean, J.-M. Jézéquel, T. Kühn, S. Mosser, H. A. Sahraoui, E. Syriani, D. Varró, and M. Weyssow, "Opportunities in intelligent modeling assistance," *Softw. Syst. Model.*, vol. 19, no. 5, pp. 1045–1053, 2020. [Online]. Available: https://doi.org/10.1007/s10270-020-00814-5

[3] T. Brown et al., "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, NeurIPS, Ed., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[4] OpenAI. [Online]. Available: https://chat.openai.com/

[5] S. Mosser, V. Reihnarz, and C. Pulgar, "Modelling Agile Backlogs as Composable Artefacts to support Developers and Product Owners," *J. Object Technol.*, 2022.

[6] F. Dalpiaz, "Requirements Data Sets (User Stories)," 2018, Mendeley Data (v1). [Online]. Available: https://data.mendeley.com/datasets/7zbk8zsd8y/1

[7] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, "Natural language processing for requirements engineering: A systematic mapping study," *ACM Comput. Surv.*, vol. 54, no. 3, apr 2021. [Online]. Available: https://doi.org/10.1145/3444689

[8] M. Weyssow, H. A. Sahraoui, and E. Syriani, "Recommending metamodel concepts during modeling activities with pre-trained language models," *Softw. Syst. Model.*, vol. 21, no. 3, pp. 1071–1089, 2022. [Online]. Available: https://doi.org/10.1007/s10270-022-00975-5

[9] D. Bajaj, A. Goel, S. Gupta, and H. Batra, "Muce: a multilingual use case model extractor using gpt-3," *International Journal of Information Technology*, vol. 14, no. 3, pp. 1543–1554, 2022.

[10] M. Cohn, *User Stories Applied: For Agile Software Development*. USA: Addison Wesley Longman Publishing Co., Inc., 2004.

[11] M. Robeer, G. Lucassen, J. M. E. M. van der Werf, F. Dalpiaz, and S. Brinkkemper, "Automated extraction of conceptual models from user stories via nlp," in *2016 IEEE 24th International Requirements Engineering Conference (RE)*, 2016, pp. 196–205.

[12] S. Arulmohan, S. Mosser, and M.-J. Meurs, "ace-design/qualified-user-stories: Version 1.0," Jul. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8136975

[13] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017, to appear.

[14] S. Mosser and S. Arulmohan, "ace-design/gpt-stories: Version 1.0," Jul. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8132676

[15] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, ser. FAccT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 610–623. [Online]. Available: https://doi.org/10.1145/3442188.3445922

[16] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data: proceedings of the 18th international conf. on machine learning, 2001," *San Francisco, CA, USA*, 2001.

[17] M. Korobov, S. Mosser, B. Mackintosh, C. D. Pietrantonio, M. E. Haase, and S. Woo, "ace-design/ace-sklearn-crfsuite: Version 0.4.1," Jul. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8132729