CONTENT 2024

The Sixteenth International Conference on Creative Content Technologies

ISBN: 978-1-68558-159-6

April 14 - 18, 2024

Venice, Italy

CONTENT 2024 Editors

Hans-Werner Sehring, NORDAKADEMIE gAG, Germany

CONTENT 2024

Forward

The Sixteenth International Conference on Creative Content Technologies (CONTENT 2024), held on April 14 – 18, 2024, continued a series of events targeting advanced concepts, solutions and applications in producing, transmitting and managing various forms of content and their combination. Multi-cast and uni-cast content distribution, content localization, on-demand or following customer profiles are common challenges for content producers and distributors. Special processing challenges occur when dealing with social, graphic content, animation, speech, voice, image, audio, data, or image contents. Advanced producing and managing mechanisms and methodologies are now embedded in current and soon-to-be solutions.

Similar to the previous edition, this event attracted excellent contributions and active participation from all over the world. We were very pleased to receive top quality contributions.

We take here the opportunity to warmly thank all the members of the CONTENT 2024 technical program committee, as well as the numerous reviewers. The creation of a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and effort to contribute to CONTENT 2024. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations and sponsors. We also gratefully thank the members of the CONTENT 2024 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope CONTENT 2024 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in the area of creative content technologies. We also hope that Venice provided a pleasant environment during the conference and everyone saved some time to enjoy this beautiful city.

CONTENT 2024 General Chair

Hans-Werner Sehring, NORDAKADEMIE - University of Applied Sciences, Germany

CONTENT 2024 Steering Committee

Raouf Hamzaoui, De Montfort University - Leicester, UK
Manfred Meyer, Westfälische Hochschule, Germany
Mu-Chun Su, National Central University, Taiwan

CONTENT 2024 Publicity Chair

José Miguel Jiménez, Universitat Politecnica de Valencia, Spain
Sandra Viciano Tudela, Universitat Politecnica de Valencia, Spain

CONTENT 2024

Committee

Table of Contents

Visual Artifacts in Software Engineering Processes

Hans-Werner Sehring
Department of Computer Science
Nordakademie
Elmshorn, Germany
e-mail: sehring@nordakademie.de

***Abstract*—Generating software solutions from (formal) models in Model-Driven Software Engineering (MDSE) approaches has many advantages, for example, model checking and traceability. Development processes that include a substantial amount of creative work rely on a manual creation of visual artifacts, and it is not feasible to work purely with formal models. As a first step towards the inclusion of graphical artifacts into MDSE, we propose two approaches: (1) Describe informal artifacts by formal model elements so that content of artifacts is related to models. (2) Generate artifacts from formal descriptions so that models become perceivable by domain experts. While the latter is not generally possible since visual artifacts are required as part of a creative process, this approach is applicable to prototyping. This way, testable designs are coherent with formal specifications.**

***Keywords*—Creative Process; Software Engineering; Software Development Processes; Model-driven Software Engineering**

I. Introduction

Designing software solutions from (formal) models and finally generating working software from these models in *Model-Driven Software Engineering* (*MDSE*) or *Model-Driven Software Development* (*MDSD*) approaches has many advantages. For example, they allow model checking of intermediate results, traceability of the results of the modeling steps [1], (partial) automation, etc.

There is class of software applications, though, that contains a substantial amount of creative work that cannot be captured by formal models for three possible reasons: (1) artifacts that result from a development step are informal in nature, for example, a graphical design sketch, (2) the work on the artifacts cannot be automated because of the creativity involved, or (3) the artifacts are preferably created using tools that do not fit into the model-driven tool chain, for example, visual prototyping tools.

In particular, graphical artifacts that provide descriptions of the solution play an important role in applications with user interaction like graphical clients, web-based systems, and mobile applications. These may be dynamic in nature. For example, prototypes exemplify the interaction with the software solution that is being developed under certain use cases and the navigation between parts of the software from a user interface perspective.

Prototypes basically provide a formal description since they consist of code. But code in itself is too expressive as to serve as a description since it cannot automatically be checked for completeness, consistency, etc.

In order to gain from MDSE advantages in creative software engineering processes, we are currently investigating means of managing visual artifacts into MDSE. To this end, model elements both serve as descriptions of visual artifacts and as formalizations that can be related to each other, checked, transformed, etc. Optimally, descriptions from creative tools can automatically be read in and related to formal descriptions, and artifacts can be updated from model elements.

In this paper, we present experiments with modeling two kinds of visual artifacts: models that can automatically be related to visual artifacts and thus co-exist with those artifacts, and models that are used to continuously generate artifacts in order to visually represent models.

We revisit model-driven as well as creative software development processes in Section II. In Section III, we introduce the Minimalistic Meta Modeling Language (M³L) as a basis for our proposal for managing visual artifacts. We study the two approaches that are proposed in this paper in Section IV. The paper concludes in Section V.

II. Artifacts in Software Engineering Processes

Software engineering processes consist of the generation of a series of artifacts, each describing the problem domain, the requirements to be addressed, or the solution.

Approaches differ in the degree of formalization of such artifacts and, therefore, in the degree of automation. Here, we specifically consider model-driven software engineering and software development processes that incorporate creative tasks in order to derive some requirements to their support.

A. Model-Driven Software Engineering

The artifacts created in *MDSE* processes are formal models that are derived from each other by means of model refinement and model transformations.

Figure 1 illustrates an MDSE process. It shows different modeling stages that correspond to different phases, here inspired by the *Model-Driven Architecture* (*MDA*) approach of the OMG [2]. A domain model represents the solution from the perspective of the subject domain, as domain concepts or requirements. In MDA, this was originally called a *Computation-Independent Model* (*CIM*; this term is not used in current specifications). It typically is an informal description, for example, done in natural language. A first formal model is a *Platform-Independent Model* (*PIM*) that gives a conceptual description of the software solution. It is transformed into a *Platform-Specific Model* (*PSM*) that adds implementation aspects. This model is used to generate a working implementation.

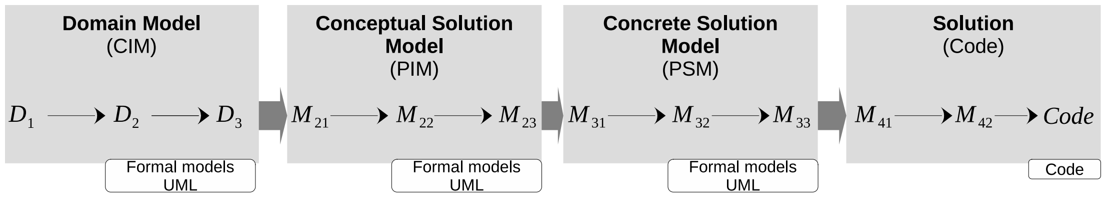

Domain Model
(CIM)
Conceptual Solution Model
(PIM)
Concrete Solution Model
(PSM)
Solution
(Code)
$D_1 \rightarrow D_2 \rightarrow D_3$
$M_{21} \rightarrow M_{22} \rightarrow M_{23}$
$M_{31} \rightarrow M_{32} \rightarrow M_{33}$
$M_{41} \rightarrow M_{42} \rightarrow Code$
Formal models UML
Formal models UML
Formal models UML
Code

Figure 1. A typical MDSE / MDSD process.

In general, we see the typical stages of domain model, conceptual software model, concrete software model, and code.

The descriptions of the PIM D_i and the models M_{ij} of the other stages in the figure indicate the (formal) artifacts created in the various stages. Within one stage, models are refined until the next stage can be started. Models of a subsequent stage provide a description from a different perspective, but they make reference to the models of the preceding stage. *Code* is a formal model from which the working solution is automatically built.

In MDSE approaches, models have to be formal in order to be subject to transformations and to reference (parts of) each other. In the MDA, for example, model transformations are specified using *Query View Transform* based on MDA's *Meta Object Facility* [3].

B. Model-Assisted Software Creation

For some kinds of software implementations, practical processes are based on artifacts that cannot fully be formalized. This depends on the application domain and is particularly true for the development of interactive applications like websites or other web-based information systems and mobile apps.

The creation of such applications incorporates creative tasks. These are typically not just amending other development tasks by adding creative input to the use and presentation of an application. On top of that, the artifacts created in creative tasks are used to communicate ideas in the course of the process, they provide a conceptual domain model from a user perspective, and they provide prototypes for testing purposes.

Creative tasks are typically manual steps that involve creativity in generating artifacts and subjective opinions in tests. As such, they do not follow a formal notation. Instead, they are often either textual or graphical descriptions. Though certain communication principles are used, they cannot be processed automatically.

Figure 2 shows typical development steps and artifacts created to model aspects of a solution. On a conceptual level, requirements are studied from the users' perspective. Resulting artifacts serve to illustrate and verify the first insights, for example, by illustrating target groups as *personas* or typical usage patterns as *customer journeys* along which users interact with a software system at different *touchpoints*. These lead to a characterization of the general solution approach, the *solution hypothesis*.

First solution aspects are created conceptually in order to validate the solution hypothesis, for example, by studying the consistency of specific parts of the solution. Feedback can be collected from users by first visual impressions, like *wireframes*, and from software developers by estimating the implementation effort, for example, based on a *catalog of graphical modules*.

More elaborate prototypes, sometimes called high-fidelity prototypes, also address visual design aspects of the solution. This way, they allow test users to get a good impression of the final software and to verify the solution approach, or to identify points that need improvement.

Based on the insights from the creative phases, the more technical considerations of software engineering follow. In processes that are based on creative activities, software aspects should be derived from the previous artifacts that were created from a user perspective. For example, the data needed at certain touchpoints and the functionality that has to be provided in specific contexts are derived from customer journeys. This is done in a phase *solution architecture* that provides an abstract specification that brings together perspectives of users, designers, software developers, and others.

C. Conceptual Software Description Artifacts

Many of the various artifacts that are generated in creative software engineering processes are informal. This means that they are not expressed in a formal language, and that they do not have formal semantics attached to them. Instead, they are targeted at human perceivers that interpret the artifacts in a subjective way.

From the many artifacts, we consider screen designs and the user interactions that lead to different presentations and interactions. These allow test users to get an impression of they way the final software will work. An artifact that serves this purpose is a *click dummy*.

Figure 3 illustrates a flow between different *dialogs* and *view states*. A dialog is a means of interaction, for example, a user interface window, a webpage, or a screen of a mobile app. A view state is one of the states a dialog can be in, for example, a search dialog in initial form, after a search has been executed, and after the search result has been refined.

Storyboards are in wide-spread use as a metaphor for such flows of dialogs [4]. These can be used in a creative fashion, but also be formalized [5].

In processes that center around the user and that incorporate creative activities, it is more common to use working software a user can interact with. There is no actual functionality underlying such software. But it gives test users the impression of working software so they can judge whether the final

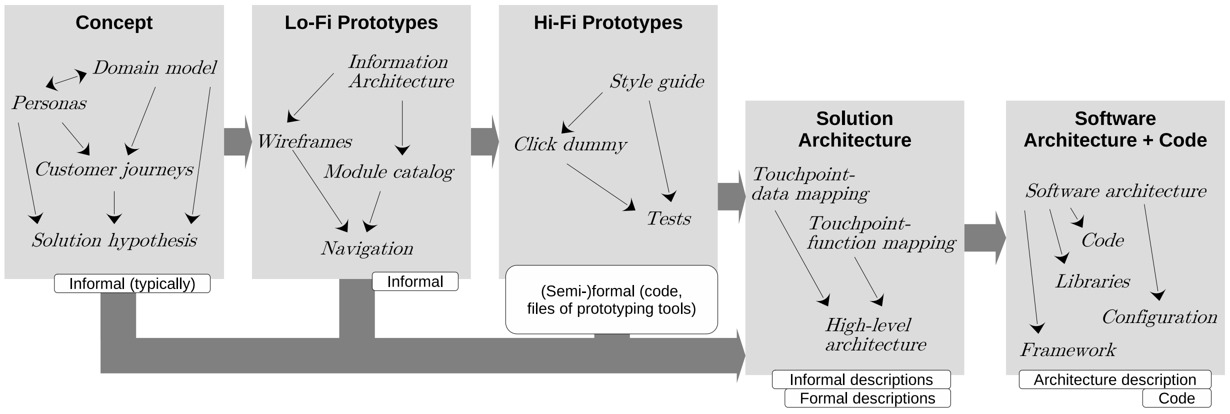


Figure 2. A typical software development process that integrates creative activities.

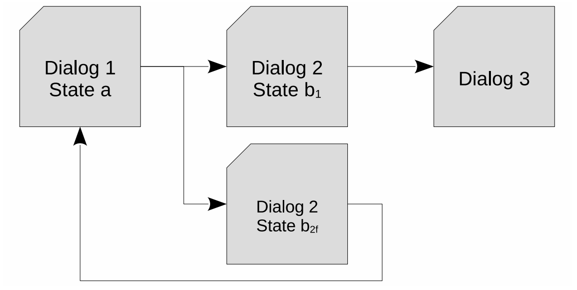


Figure 3. A sample flow of dialogs and dialog states.

software works as expected [6]. To this end, it simulates changes between dialogs and changing the status of a dialog.

In practice, such shallow prototypes are created as configurations in prototyping software tools. In such cases, the prototypes do not evolve into working software. They are maintained as singular artifacts that can later be analyzed.

In an MDSE scenario, prototypes can be generated from evolving models, though. In this case, they directly reflect the current modeling state. Models can be updated based on user feedback, and then a new revision of the prototype can be generated. This way, work on the client does not only lead to improved prototypes, but it directly leads to an improved model.

III. Minimalistic Meta Modeling Language (M³L)

For the discussion in this paper, we use the M³L since it proved to be a suitable language for the modeling of content of various artifacts. To this end, we briefly introduce the M³L, and we present some exemplary base models for the representation of artifacts.

A. Basic M³L Constructs

In this section, we briefly introduce the M³L by highlighting those features that are central to the underlying experiments.

The basic M³L statements are:

- **`A`**: the declaration of or reference to a *concept* named *A*
- **`A`** `is a` **`B`**: refinement of a concept *B* to a concept *A*. *A* is a *specialization* of *B*, *B* is a *generalization* of *A*.
- **`A`** `is a` **`B`** `{` **`C`** `}`: containment of concepts. *C* belongs to the *content* of *A*, *A* is the *context* of *C*.
- **`A`** `|=` **`D`**: the *semantic rule* of a concept. Whenever *A* is referenced, actually *D* is bound. If *D* does not exist, it is created in the same context as *A*.
- **`A`** `|-` **`E F G .`**: the *syntactic rule* of a concept that defines how a string is produced from a concept, respectively how a concept is recognized from a string. When the representation of A is requested, it is produced by a concatenation of the strings produced out of E, F, and G. When no syntactic rule is defined, a concept is represented by its name. Vice versa, an input that constitutes the name of a concept without a syntactic rule leads to that concept being recognized.

If a concept that is referenced by one of the statements exists or if some other concept evaluates (see below) to the reference, then this one is bound. Otherwise, the concept is created as defined by the statement.

Existing concepts can be redefined. For example, with the definitions above, a statement

`A` `is an` **`H`** `{` **`C`** `is the` **`I`** `}`

redefines A to have another generalization H and C (in the context of A) to be the only specialization of I.

Every context constitutes a *scope*. A redefinition of a concept in a context is only applied in that context. When a redefinition of a concept takes place in another context as the original definition, we call that redefinition a *derivation*.

Scopes also define *visibility*. A concept is visible in the context it is (re-) defined in, and all contained contexts. Concepts from foreign contexts can be made visible with

`ForeignConcept` `from` **`ForeignContext`**

B. Concept Evaluation

The concepts that are defined by such statements are *evaluated* when used. Evaluation means looking up or creating concepts and applying semantic rules. This is done in the following form:

1) Evaluation is performed with respect to a context from which concept definitions are taken.

2) The effective definition of a concept in some context is the set of all definitions in that context and its surrounding base contexts (transitive).
3) A concept (transitively) inherits base concepts and content from its base concepts
4) A concept A is a derived base concept of a concept B if B *matches* A. Matching is defined as:
   - A and B share a common base concept (or A has none).
   - For every content C of A there exists content D of B where D *matches* C.
5) Let EC_A be the union of all base concepts and all derived base concepts of a concept A. If a concept in EC_A has a defined or inherited semantic rule, then the result of this rule defines candidates for the evaluation of A. If no concept in EC_A carries a semantic rule, then the whole set forms the set of candidates.
6) Finally, all candidate concepts are *narrowed down*: for each concept, the most specific matching subconcept is added to the result.

C. M³L Example

For a simple modeling example, consider the following M³L statements from the area of programming language design:

```
Boolean is a Type
Statement
IfThenElse is a Statement {
 Condition is a Boolean
 ThenStatement is a Statement
 ElseStatement is a Statement }
IfTrue is an IfThenElse {
 True is the Condition } |= ThenStatement
IfFalse is an IfThenElse {
 False is the Condition }|= ElseStatement
```

These describe the behavior of a conditional statement as it is found in typical imperative programming languages.

Adding an additional statement in order to provide a concrete program,

```
MyCond {
 ConceptEvaluatingToBool is the Condition
 SomeStatement is the ThenStatement
 SomeOtherStatement is the ElseStatement}
```

leads to the following evaluation: depending on the evaluation result of *ConceptEvaluatingToBool*, either *IfTrue* or *IfFalse* is a derived base concept of *MyCond* (shared base concept *Statement* and matching Content *True*). Thus, *MyCond* inherits the semantic rule from this derived base concept, making it evaluate to the right conditional branch. Candidates are either *ThenStatement* or *ElseStatement* that are finally narrowed down to *SomeStatement* or *SomeOtherStatement*, respectively.

IV. Managing Models of Visual Artifacts

To allow some of the creative artifacts to be integrated into a model-driven development process, we propose to define models that reflect the content of those artifacts. These models allow handling the representations in model transformation processes. This way, for example, traceability between models and thus between artifacts is achieved. It can be made explicit, which artifacts provided input to which other artifacts.

We see two basic approaches that are handled by the following two subsections: artifacts that are entities in their own right where models are representing the current state of the artifacts, and artifacts that are generated from models that provide one aspect of the development process.

A. Co-existence of Artifacts and Models

The M³L is universal and has many applications. Among other modeling tasks, it has proven useful for describing content and creating documents from content [7]. This applies both to content models, as well as content items since the M³L does not distinguish model layers, such as type and instance.

Using the M³L to define a content management structure, the content of artifacts can be maintained in models and be represented by documents generated out of the models. Changes to documents can be re-read in order to update the content concepts.

For an example of structured content, with a content model like:

```
UserStory is a Content {
 Title is a String
 Text is a FormattedString
 StoryPoints is a FibonacciNumber }
```

according content can be created:

```
UserStory123 is a UserStory {
 "As a sales person I ..." is the Title
 "When a sales person ..." is the Text
 8 is the StoryPoints }
```

For presentation purposes, content is added to document descriptions. For example, when creating text documents, there might be definitions like:

```
PublishedDocPart { Content }
UserStoryDetailPage is a PublishedDocPart{
 Content is a UserStory }
WebPage is a PublishedDocPart {
 Title is a String }
PrintDocumentPage is a PublishedDocPart {
 PageNumber is a Number }
```

For textual formats, like HTML and JSON, documents can be rendered from concepts through syntactic rules of content as introduced in the previous section. On the level of a content model, syntactic rules describe document templates, on the content item level they render single document instances.

For the sample content definitions above, an HTML template may look like outlined in Figure 4. In this example, there are some basic definitions for HTML code generation (or code parsing) in the context *HTML*. These include the redefinition of the concept *WebPage* from above with a syntactic rule for HTML generation. The sample code sketches a simple skeleton of HTML code.

```
HTML {
 <html> </html> <head> </head>
 <title> </title> <body> </body> ...
 WebPage
 |- <html>
     <head> <title> Title </title> </head>
     <body> Content </body>
    </html> . }
UserStoryHTML is an HTML {
 UserStoryDetailPage {
  Title is the Title from Content
  Content {
   Title |- "<h1>" Title "</h1>" .
   Text |- "<p>" Text "</p>" .
   StoryPoints |- "SP: " StoryPoints .
  } |- Title StoryPoints Text . }
```

Figure 4. Sketch of an HTML publication model.

Also, some tags for HTML elements are defined to indicate that several basic definitions will be provided here. Please note that, e.g., *<html>* is a valid concept name. Since new concepts are declared the first time they are referenced, and because they syntactically evaluate to their name by default, they can also be used in all syntactic rules like string literals.

The particular HTML generation of representations of user stories is defined in a subcontext *UserStoryHTML*. Here, the different parts of a user story's content are equipped with syntactic rules that produce (or recognize) page fragments.

HTML for a webpage that represents a user story is generated by creating a *UserStoryDetailPage* with a specific user story as content:

```
MyUserStoryPage is a UserStoryDetailPage {
 SomeUserStory is the Content }
```

The syntactic rule of *WebPage* from *HTML* is inherited by *UserStoryDetailPage*. Therefore, the page skeleton will be generated, and *Content* is embedded. For *Content*, a syntactic rule is defined in the context of *UserStoryDetailPage* that triggers the syntactic rule of the user story's content.

Manual changes to an HTML page can be re-read by the syntactic rules, leading to the M³L concepts being updated.

B. Generation of Artifacts from Models

Artifacts can be generated from models in many cases, but changes to the artifacts cannot automatically be re-engineered to models. In such scenarios, we can use "micro-MDSE" processes that drive the creation of but one artifact. Eventually, several such processes are running for co-evolving artifacts.

As an example, we consider click dummies in this section. Click dummies are shallow prototypes that give an impression of the final software product by offering a consistent part of a User Interface (UI) to test users. When click dummies are generated from models, any changes to the UI because of user feedback will be applied by reworking those models. This way, UI designs become manageable in MDSE processes.

```
UIElement { Id }
VisibleUIElement is a UIElement {
 Dimension is a ...
 ... }
Container is a UIElement {
 Components is a UIElement }
TextField is a VisibleUIElement {
 Text is a String }
```

Figure 5. Sketch of a base model of static UI elements.

```
ActivatableUIElement is a UIElement {
 ActionHandler }
Button is a VisibleUIElement,
          an ActivatableUIElement {
 Label is a String
 Icon is an Image }
ActionHandler {
 Condition is a Boolean
 Effect }
ExecutedActionHandler {
 True is the Condition
} |= Effect
UIEvent {
 Target is an ActivatableUIElement
} |= ActionHandler from Target {
       True is the Condition }
   |- "{" "\"target\"" ":"
          "\""Id from Target"\"" "}"
ClickEvent is a UIEvent
```

Figure 6. Sketch of a base model of dynamic UI elements.

The considerations of model-based UI generation are based on earlier work on the topic [8]. In the meantime, quite some related work emerged [9].

1) Base UI Model: Static aspects of a UI consist of a definition of UI components and possible combinations of them. Figure 5 outlines an example.

UIElement is a base definition of all UI components. Visible UI elements are categorized as *VisibleUIElement* that adds some parameters required to draw a component. A *Container* is an example of an invisible component that defines a UI layout.

To relate input to components, we assume that every UI component carries an ID. Alternative approaches might, for example, use a component's path in the UI layout hierarchy.

2) Dynamic UI Aspects: To define the dynamic behavior of a UI, user interactions have to be modeled. A UI description needs a runtime environment that visualizes the defined UI elements, and that handles user input. For an example, assume a runtime environment that hands over events as JSON strings, for example, for mouse clicks, and that receives UI updates by marshaled/HTML components.

Events are handled by *ActionHandler*s as lined out in Figure 6. Dynamic UI components are *ActivatableComponent*s

```
SearchDialog is a UIModel {
 SearchView is a Panel {
  1 is the Id
  QueryField is a TextField
  SearchButton is a Button {
   2 is the Id
   Search is the Label }
  ...
} } |- SearchView
SearchDialogInitial is a SearchDialog {
 SearchView {
  SearchButton {
   Action1 is an ActionHandler {
    "manyres" is the Text from QueryField
    SearchDialogManyResults is the Effect}
   Action2 is an ActionHandler {
    "fewres" is the Text from QueryField
    SearchDialogFewResults is the Effect }
} } }
SearchDialogManyResults is a SearchDialog{
 ... }
SearchDialogFewResults is a SearchDialog {
 ... }
```

Figure 7. Example of views and view transitions.

that may have a handler. A handler that shall execute its action is activated by its *Condition* set to true, making it become a derived subconcept of *ExecutedActionHandler* and its *Effect* to be the result.

JSON messages from the environment are parsed using the syntactic rule of *UIEvent*. The *Id* of a resulting *UIEvent*'s *Target* is set to the ID contained in the JSON message. As a consequence, *Target* evaluates to the actual target component and the target's *Action* is activated.

3) Example of a User Interaction Model: Figure 7 shows a sample application of such a UI model. When an event is sent to the *SearchButton*, the *Condition* of all action handlers, here *Action1* and *Action2*, is set to *True*. When evaluating *ActionHandler* in the semantic rule of *UIEvent*, it evaluates to those actions for whom also the text of the query field matches. This way, different paths through the sequence of dialogs are selected by test users. If none is applicable, the event just evaluates to the target's base *ActionHandler* that has no defined *Effect*. The *Effect* of the search actions (*Action1* and *Action2*) is a new *SearchDialog*, that "replaces" the current one. The assumed surrounding runtime environment is responsible for rendering the new search dialog through the syntactic rule defined for *SearchDialog*.

V. Conclusion

We conclude with a summary and an outlook.

A. Summary

We presented two approaches to the management of informal artifacts by formal modeling elements. The first experiment demonstrates the possibility to generate textual descriptions from abstract descriptions by using typical content management functionality.

Prototypical implementations can be generated the same way as the final software solution. A second experiment showed that this can be done in the same modeling framework as the first experiment.

Therefore, potentially different ways of including creative work can be combined. For example, a graphical design specification can be created using the first approach of co-evolving documentation and models, and can finally be used to related to a prototype description on the level of models.

B. Outlook

Additional research is need to fully investigate round-trip engineering that consists of alternating (manual) creative and (automated) modeling steps that operate on the same set of artifacts.

Implementation work, for example extending the M³L, is required to include other formats of visual artifacts into the process so that models do not prescribe the form of those artifacts. With the incorporation of such formats, practical processes in which proprietary design tools are used can be subject to further experiments.

Acknowledgment

The author thanks NORDAKADEMIE gAG for funding of this work. Various colleagues from the fields of user experience design and software engineering helped define the initial experiments.

References

[1] I. Galvao and A. Goknil, "Survey of Traceability Approaches in Model-Driven Engineering," Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, 2007, pp. 313-313.
[2] Object Management Group. *Model Driven Architecture (MDA)*, MDA Guide rev. 2.0, OMG Document ormsc/2014-06-01, [Online] Available from: https://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf. 2024.2.10.
[3] Object Management Group. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*, OMG Document Number formal/2008-04-03, [Online] Available from: https://www.omg.org/spec/QVT/1.0/PDF/. 2024.2.10.
[4] C. van der Lelie, "The value of storyboards in the product design process," Personal and Ubiquitous Computing, volume 10, pp. 159–162, 2006.
[5] K.-D. Schewe and B. Thalheim, "Storyboarding," Design and Development of Web Information Systems, Springer, pp. 61–109, 2019.
[6] S. Böhm and S. Graser, "AI-based Mobile App Prototyping: Status Quo, Perspectives and Preliminary Insights from Experimental Case Studies," Proceedings Sixteenth International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services, ThinkMind, 2023, pp. 29–37.
[7] H.-W. Sehring, "On Integrated Models for Coherent Content Management and Document Dissemination," Proceedings of the Thirteenth International Conference on Creative Content Technologies, ThinkMind, 2021, pp. 6–11.
[8] H.-W. Sehring, "Adaptable and adaptive visualizations in concept-oriented content management systems," International Journal on Advances in Software, volume 3, numbers 1 and 2, ThinkMind, pp. 265–279, 2010.
[9] J. C. Mejias, N. Silega, M. Noguera, Y. I. Rogozov, and V. S. Lapshin, "Model-Driven User Interface Development: A Systematic Mapping," Proceedings of the 8th Iberoamerican Workshop on Human-Computer Interaction, Springer, 2022, pp. 114–129.

Automatic Semantic Image Tagging at Scale: AI-Powered Command-Line Tool Based on CLIP

Yurij Mikhalevich
Lightning AI
Dubai, United Arab Emirates
email: yurij@mikhalevi.ch

***Abstract*—This study introduces an efficient and scalable solution for image tagging through the utilization of OpenAI's Contrastive Language-Image Pre-Training (CLIP) model. It features a user-friendly Command-Line Interface (CLI) and leverages a caching mechanism for image features introduced by the rclip tool, which is powered by the SQLite 3 Relational DataBase Management System (RDBMS). This enables effective tagging of image catalogs already indexed by rclip. The performance of this method was tested on the ObjectNet dataset, achieving 27.22% accuracy for the top-1 prediction and 50.42% for the top-5 predictions. A scalability analysis revealed that both the indexing and tagging processes increase linearly with the image count. For instance, processing 50,273 unindexed images on an Apple M1 Max CPU was 31.66 times longer than tagging 965 unindexed images, and handling 50,273 indexed images was 31.15 times longer than 965 indexed images. This strategy is particularly beneficial for sectors that handle large amounts of visual content and require external processing tools, such as the media and entertainment, security, and healthcare industries.**

Keywords-image tagging; image indexing; photo management; computer vision

I. Introduction

The unveiling of the CLIP model by OpenAI in 2021 has captured widespread interest across the domains of Natural Language Processing (NLP) and Computer Vision (CV). This innovative model is capable of understanding advanced image representations by analyzing a vast collection of 400 million image and text pairs sourced from the internet. CLIP uniquely identifies the most applicable text snippet for an image through natural language guidance without being directly trained for such tasks. Its zero-shot learning capabilities mirror those found in GPT-2 and GPT-3 [1]. The authors of CLIP have showcased its ability to match the performance of the original ResNet50 on the ImageNet dataset in a zero-shot setting without using any of the 1.28 million labeled examples. This breakthrough addresses some of the most daunting challenges in the field of computer vision.

Given its capabilities, CLIP emerges as an ideal foundation for developing a semantic image tagging tool capable of operating in a zero-shot manner with any user-provided tags or images. This article explores this particular application of CLIP.

The rapid expansion of data, particularly image data, underscores the timeliness and significance of this research. The surge in digital device usage and internet accessibility has led to an unprecedented increase in image production and sharing online, complicating the task of manually locating specific images. In a previous study, we introduced a method that employs natural language queries for image searches using the CLIP model, resulting in the development of a tool named `rclip` [2]. While `rclip` has broad applications, there are a lot of other use cases where it's important to be able to tag images to enable their manual cataloging and search using 3rd-party software. This makes an efficient image tagging solution increasingly necessary and relevant.

In Section 2, the paper explores related works in the field of image tagging. Section 3 describes the proposed method for image tagging, which is built on top of `rclip` powered by CLIP. Section 4 presents the implementation details of the proposed method. Section 5 discusses the performance of the proposed method. Section 6 presents the tagging quality measurement results of the proposed method. Section 6 discusses the implementation and future plans. Finally, Section 8 concludes the paper and discusses future work.

II. Related Works

The advancement of NLP and CV has been significantly influenced by the development of models capable of understanding and generating insights from both text and images. The CLIP model by OpenAI marks a pivotal moment in this journey, leveraging a novel approach to learn visual concepts from natural language descriptions. This section reviews the literature that contextualizes CLIP within the broader field of image tagging and multimodal learning.

One of the earliest attempts to bridge the gap between vision and language is the work by Socher et al. [3], who introduced a model for generating descriptions of image contents, paving the way for subsequent research in multimodal learning. Their model was among the first to use a neural network to combine visual and textual data, setting a foundation for future exploration in the field.

The concept of using neural networks for image classification was revolutionized by Krizhevsky et al. [4], with the introduction of AlexNet, which demonstrated the potential of deep learning in computer vision tasks. This work was instrumental in showing how deep learning could be applied to achieve significant improvements in image classification accuracy.

Following this, the development of the ResNet model by He et al. [5] represented another major step forward, introducing the concept of residual learning to enable the training of much deeper networks. The advancements in image classification models like ResNet laid the groundwork for more sophisticated image understanding and tagging capabilities.

In parallel to improvements in image classification, research in NLP was making strides with the development of models like BERT by Devlin et al. [6], which introduced a new method for pre-training language representations. BERT's success in understanding context and nuance in text provided inspiration for exploring similar pre-training approaches in multimodal models.

The intersection of NLP and CV began to take a more defined shape with the introduction of models like ViLBERT by Lu et al. [7], which employed separate pathways for processing visual and textual information before integrating them, demonstrating improved performance on tasks requiring both visual and textual understanding.

The development of CLIP by Radford et al. [1] stands as a significant milestone, combining the learnings from both fields to create a model that learns visual concepts from natural language descriptions. This model's ability to perform zero-shot image classification and tagging by leveraging a vast dataset of image-text pairs collected from the internet has opened new avenues for research and application.

In the domain of image tagging specifically, prior works have explored various approaches for automating the process. Weyand et al. [8] introduced a method for geolocating images using a deep neural network, demonstrating the potential of using image content to infer metadata. This work highlighted the capability of deep learning models to extract and infer contextual information from images beyond simple object recognition.

The advancement of deep learning in image tagging has been notably propelled by research focusing on the intricacies of multi-label classification. One important study by Wang et al. [9] introduced a CNN-RNN framework that enhances the accuracy of tagging by leveraging the correlations between labels in a multi-label setting. Their method demonstrates how models can be effectively trained to predict multiple relevant tags for a single image, considering the dependencies among these tags to improve the precision of the tagging process. This approach underlines the complexity of image tagging as a multifaceted problem, benefiting from models that capture and utilize the nuanced interrelations among tags to produce more accurate and contextually relevant annotations. The progress highlighted by Wang et al. underscores the transformative potential of deep learning techniques in the domain of semantic image tagging, offering insights into the development of more sophisticated and efficient tagging solutions.

Subsequently, the integration of attention mechanisms, as seen in models like the Transformer [10], has been adapted to multimodal learning, enhancing the ability of models to focus on relevant parts of the input when generating tags or descriptions. This approach has notably improved the precision of image tagging, facilitating more nuanced interpretation of visual and textual data. Among such models, ViLBERT [7] and CLIP [1] exemplify the advancements in multimodal learning. ViLBERT utilizes a co-attentional layer to process visual and textual inputs in parallel, enabling improved performance on tasks like visual question answering and visual commonsense reasoning. Similarly, CLIP leverages transformer architectures to learn visual concepts from natural language supervision, achieving a profound understanding of images through textual descriptions. These models underscore the significant role of transformers in bridging the semantic gap between different types of data, thereby enhancing AI's capability in multimodal understanding and interaction.

The exploration of CLIP [1] and its application to image tagging, as discussed in this paper, builds upon these foundational works. By leveraging CLIP's zero-shot learning capabilities, we propose a novel method for semantic image tagging that can adapt to a wide range of tags and images, addressing the challenges posed by the ever-increasing volume of digital images.

III. Method

CLIP's text transformer allows us to convert a text label (tag) into a n-dimensional vector (where n differs depending on the CLIP model used). With CLIP's image transformer, it is possible to convert an image to a n-dimensional vector. Then, we can calculate the dot product (Equation 1) of the normalized image vector and each of the normalized tag vectors. After this, we pick tags with the dot product higher than the configured threshold; this gets us the tags that match the image the most.

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \tag{1}$$

To allow tagging any kind of image with any set of tags, the approach suggests allowing the input of a custom list of tags. While this approach works reasonably fast on a few images and tags, it may not be scalable when dealing with a large number of images or tags. This is particularly challenging if there is no access to GPUs to run the CLIP model. In order to address this scalability issue, this article proposes two optimizations:

- pre-compute tag feature vectors and reuse them when processing all of the images;
- utilize the `rclip`'s [11] cache layer to avoid computing feature vectors for images already processed by `rclip` and allow us to retag the images quickly.

The cache layer implemented by `rclip` stores the computed image vectors on disk after the initial processing of images so that they can be accessed later. We propose to reuse this logic in the image tagging process. This approach reduces the computational overhead associated with image tagging and improves the response time for users. This is particularly useful when the user needs to retag the images with a different set of tags or when the user already has their images indexed by `rclip`.

`rclip` also handles adding new images to the cache to avoid recomputing the whole cache when the image catalog is updated.

Once we obtain the tags, we store them in the image metadata. This allows other software to utilize the tags. For example, the user can use the tags to search for images using the operating system's file manager or a third-party image management software.

IV. Implementation

The method described above is implemented in the Python utility called rtag [12]. rtag provides an easy-to-use CLI interface that allows users to tag images within any directory on a computer where rtag is installed. Tagging images within nested directories of a complex directory subtree is also supported. To use it, the user should open the terminal, navigate to the directory containing the files that they want to tag, type rtag, and hit “Enter.”

The solution uses the rclip library [11] to compute the feature vectors. rclip uses ViT-B/32 version of the OpenAI’s CLIP model [1]. The tool writes the computed tags to the image IPTC metadata [13] using the IPTCInfo3 Python library [14]. A simplified version of the image processing loop source code is shown in Figure 1.

By default, rtag uses the list of the ImageNet-1k [15] labels as tags. The user can provide their own list of tags by specifying the --tags-filepath argument when running rtag. For example, rtag --tags-filepath ./path/to/tags.txt. ./path/to/tags.txt should contain a newline-separated list of tags. Figure 2 shows how the tags are loaded from the file.

rtag supports two modes of operation: append and overwrite. In the append mode, the tool appends the computed tags to the existing tags in the image metadata. In the overwrite mode, the tool overwrites the existing tags with the computed tags. The user can specify the mode by providing the --mode argument when running rtag. For example, rtag --mode append. Figure 1 shows how setting the mode changes the tool behavior.

rtag also supports the --threshold argument, which allows the user to specify the similarity threshold. The user can use this argument to control the number of tags that are written to the image metadata. The default value is 0.25, which produces optimal results.

Another useful rtag feature is the ability to do a dry run. The user can use the --dry-run argument to see what tags would be written to the image metadata without actually writing them. This is particularly useful when the user wants to see how the tool behaves with a specific set of tags and images and wants to adjust the similarity threshold.

The rtag source code is published on GitHub under the MIT license [12].

V. Performance

rtag was benchmarked on the ObjectNet 50,273 images dataset [16], on the Apple M1 Max CPU. Table I shows how rtag performs when tagging previously indexed and unindexed images. As you can see from the table, running rtag on 50,273 unindexed images took 6m21.250s, while tagging 965 unindexed images took 0m12.040s. A similar linear scaling relationship is preserved when running rtag on indexed images. Tagging 50,273 indexed images took 4m44.790s while tagging 965 indexed images took 0m09.140s.

```
tag_features = (
  model.compute_text_features(tags)
)

for image in tqdm(
  rclipDB.get_image_vectors_by_dir_path(
      current_directory),
  unit='images',
):
  image_path = image['filepath']
  image_features = np.frombuffer(
    image['vector'],
    np.float32,
  )

  similarities = image_features @
      tag_features.T

  new_tags = []
  for tag, similarity in zip(tags,
      similarities):
    if similarity > args.threshold:
      new_tags.append(tag)

  if not new_tags:
    continue

  image_metadata = IPTCInfo(
    image_path,
    force=True,
  )

  if args.mode == 'append':
    existing_tags = image_metadata['
        keywords']
    image_metadata['keywords'] = [*set(
        existing_tags + new_tags)]
  elif args.mode == 'overwrite':
    image_metadata['keywords'] = new_tags
  else:
    raise ValueError(f'Invalid■mode:■{
        args.mode}')

  image_metadata.save()
```

Figure 1. Image processing loop

```
def load_tags_from_file(path: str):
  with open(path, 'r') as f:
    return f.read().splitlines()

tags_filepath = args.tags_filepath or
    get_imagenet_tags_filepath()
tags = load_tags_from_file(tags_filepath)
```

Figure 2. Tags loading

It should be noted that before running the benchmarks, the dataset was resized to 224px by smaller dimension and converted to JPG using ImageMagick [17].

The real-life `rtag` application possibilities go far beyond results demonstrated in these benchmarks because `rtag` can be used with any tags and images and not just the ones present in the dataset.

VI. Tagging Quality

As Table II shows, `rtag` achieves 27.22% top-1 accuracy and 50.42% top-5 accuracy rate on the `ObjectNet` [16] 50,273 images dataset. The `ObjectNet` dataset was chosen for the quality measurement because it is a diverse and challenging dataset that contains images of objects in their natural environments.

The table also shows that `rtag` shows better results when we compute label feature vectors from a `photo of [tag]` string instead of using the `[tag]` as is. Further research is needed to understand whether it is a good idea to default to using `photo of [tag]` in `rtag`.

The benchmark source code is available in the project repository on GitHub [12].

To get a better understanding of `rtag`'s performance, see Figure 3, Figure 4, and Figure 5 showing a few images from the `ObjectNet` dataset tagged by `rtag`.

TABLE I
Tagging performance on Apple M1 Max CPU

Dataset	# of images	Indexed	Processing time
ObjectNet dataset	50,273	No	6m21.250s
ObjectNet dataset	50,273	Yes	4m44.790s
ObjectNet subset	965	No	0m12.040s
ObjectNet subset	965	Yes	0m09.140s

TABLE II
rtag tagging quality

Model	Top-1 accuracy	Top-5 accuracy
ObjectNet	25.28%	48.05%
ObjectNet photo of [tag]	27.22%	50.42%

VII. Discussion

ObjectNet is a large, real-world, challenging test set for object recognition with control where object backgrounds, rotations, and imaging viewpoints are random. Despite this making the test set perfect for benchmarking `rtag`, it would be interesting to see how `rtag` performs on other datasets.

The development of the command-line tool, `rtag`, which is based on `rclip` and employs OpenAI's CLIP model, has resulted in an efficient and user-friendly utility for image tagging. However, there are still possibilities for further improvements.

Future plans include:

- to improve the tool's tagging performance by processing multiple images in parallel;
- to add support for writing tags into the XMP sidecar files and leaving the image files intact;
- to improve the tool's performance by preventing it from re-indexing files when they are renamed;
- to assess the tool's performance on other datasets;
- to enrich `rtag`'s tagging capabilities by utilizing metadata, which already exists within the images, e.g., existing GPS coordinates can be used to tag images with human-readable location-based tags;
- to do an in-depth comparison of `rtag` with other existing image tagging tools.

Even with its current performance and capabilities, `rtag` is an incredibly valuable tool.

VIII. Conclusion

In summary, this paper introduces a practical and scalable method of tagging images of any kind with any set of labels. The method is based on the CLIP model. The approach has demonstrated impressive results on the ObjectNet dataset, indicating its potential applicability to a wide range of industries reliant on visual data and using image processing tools requiring images to be tagged.

References

[1] A. Radford *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*, PMLR, 2021, pp. 8748–8763.

[2] Y. Mikhalevich, "Using text queries to look up unlabeled images: A command-line search tool based on clip," in *Proceedings of CONTENT 2023, The Fifteenth International Conference on Creative Content Technologies*, IARIA, 2023, pp. 7–11.

[3] R. Socher, A. Karpathy, Q. V. Le, C. D. Manning, and A. Y. Ng, "Grounded compositional semantics for finding and describing images with sentences," *Transactions of the Association for Computational Linguistics*, vol. 2, pp. 207–218, 2014.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[7] J. Lu, D. Batra, D. Parikh, and S. Lee, "Vilbert: Pre-training task-agnostic visiolinguistic representations for vision-and-language tasks," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2019, pp. 13–23.

[8] T. Weyand, I. Kostrikov, and J. Philbin, "Planet - photo geolocation with convolutional neural networks," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*, Springer, 2016, pp. 37–55.

[9] J. Wang *et al.*, "Cnn-rnn: A unified framework for multi-label image classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2285–2294.

[10] A. Vaswani *et al.*, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 6000–6010.

[11] Y. Mikhalevich, *rclip*, version 1.7.26, Feb. 2024. [Online]. Available: https://github.com/yurijmikhalevich/rclip [retrieved: Feb. 2024].

[12] Y. Mikhalevich, *rtag*, version 0.1.0, Feb. 2024. [Online]. Available: https://github.com/yurijmikhalevich/rtag [retrieved: Feb. 2024].

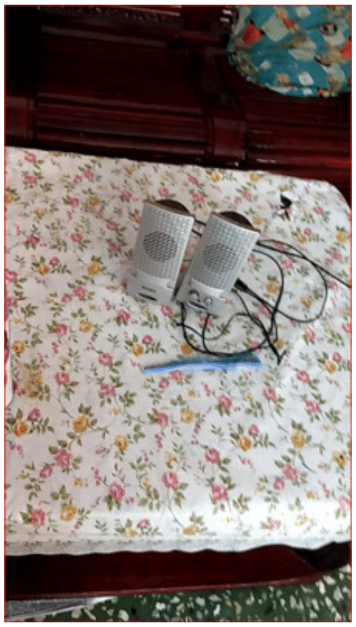
[13] *Iptc photo metadata standard*, Jan. 2023. [Online]. Available: https://www.iptc.org/standards/photo-metadata/iptc-standard/ [retrieved: Feb. 2024].

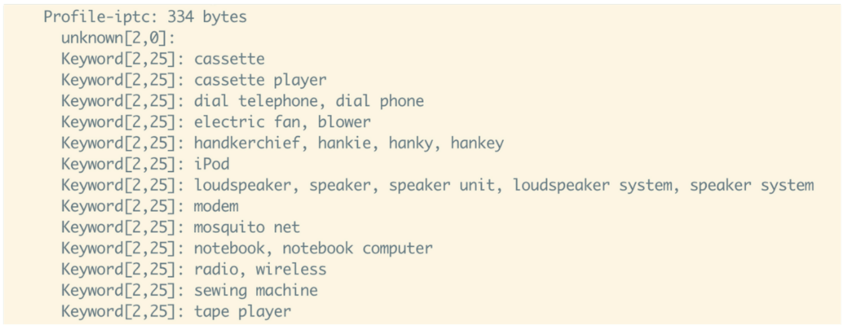
[14] T. Gulacsi and J. Campbell, *IPTCInfo3*, version 2.1.4, 2019. [Online]. Available: https://github.com/james-see/iptcinfo3 [retrieved: Jan. 2024].

[15] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[16] A. Barbu *et al.*, "Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models," vol. 32, 2019, pp. 9448–9458.

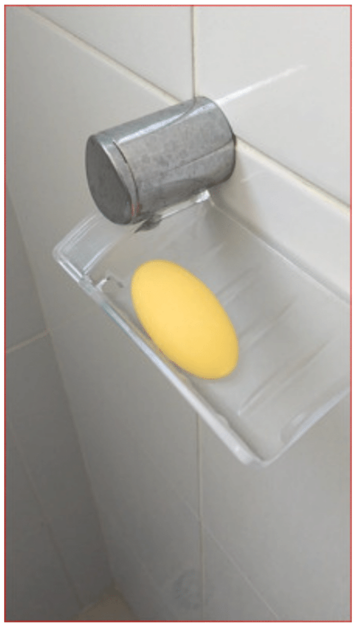
[17] ImageMagick Studio LLC, *Imagemagick*, version 7.0.10, Jan. 4, 2023. [Online]. Available: https://imagemagick.org [retrieved: Feb. 2024].

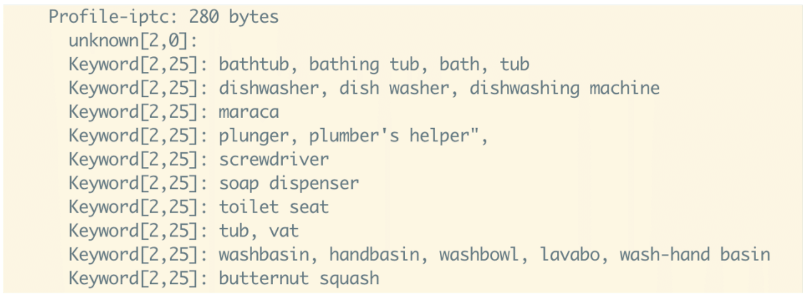




```
Profile-iptc: 334 bytes
  unknown[2,0]:
  Keyword[2,25]: cassette
  Keyword[2,25]: cassette player
  Keyword[2,25]: dial telephone, dial phone
  Keyword[2,25]: electric fan, blower
  Keyword[2,25]: handkerchief, hankie, hanky, hankey
  Keyword[2,25]: iPod
  Keyword[2,25]: loudspeaker, speaker, speaker unit, loudspeaker system, speaker system
  Keyword[2,25]: modem
  Keyword[2,25]: mosquito net
  Keyword[2,25]: notebook, notebook computer
  Keyword[2,25]: radio, wireless
  Keyword[2,25]: sewing machine
  Keyword[2,25]: tape player
```

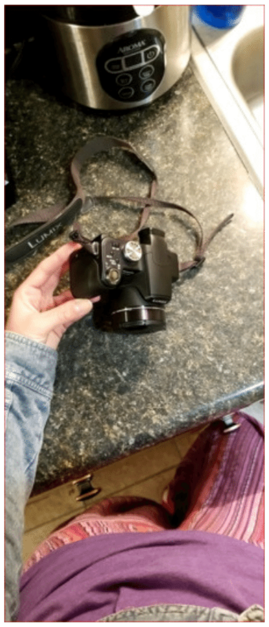
Figure 3. An image of speakers tagged by rtag

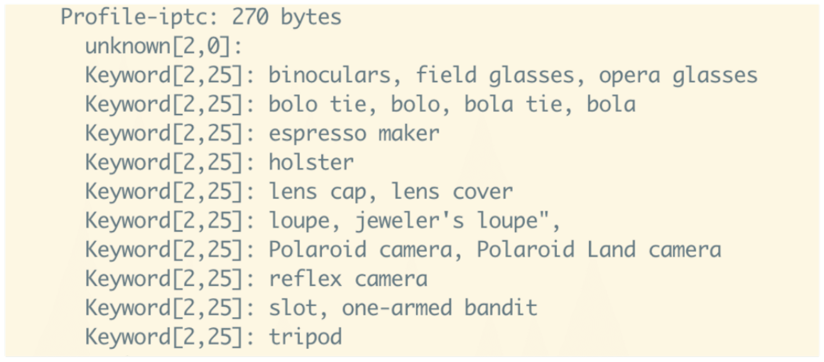




```
Profile-iptc: 280 bytes
  unknown[2,0]:
  Keyword[2,25]: bathtub, bathing tub, bath, tub
  Keyword[2,25]: dishwasher, dish washer, dishwashing machine
  Keyword[2,25]: maraca
  Keyword[2,25]: plunger, plumber's helper",
  Keyword[2,25]: screwdriver
  Keyword[2,25]: soap dispenser
  Keyword[2,25]: toilet seat
  Keyword[2,25]: tub, vat
  Keyword[2,25]: washbasin, handbasin, washbowl, lavabo, wash-hand basin
  Keyword[2,25]: butternut squash
```

Figure 4. An image of a soap tagged by rtag





```
    Profile-iptc: 270 bytes
      unknown[2,0]:
      Keyword[2,25]: binoculars, field glasses, opera glasses
      Keyword[2,25]: bolo tie, bolo, bola tie, bola
      Keyword[2,25]: espresso maker
      Keyword[2,25]: holster
      Keyword[2,25]: lens cap, lens cover
      Keyword[2,25]: loupe, jeweler's loupe",
      Keyword[2,25]: Polaroid camera, Polaroid Land camera
      Keyword[2,25]: reflex camera
      Keyword[2,25]: slot, one-armed bandit
      Keyword[2,25]: tripod
```

Figure 5. An image of a camera tagged by rtag

Automatic Generation of Illustrations for Children's Fairy Tales From Text: AI-Powered App Based on DALL-E 3

Irina Kogai
Dubai, United Arab Emirates
email: ikogai.dev@gmail.com

Abstract—**This paper explores the capabilities of DALL-E 3, an advanced artificial intelligence model, to automatically generate illustrations for children's fairy tales. The primary objective is to assess the relevance of the images produced by this Artificial Intelligence (AI), examining how well these illustrations can complement and enhance the storytelling experience for young audiences. To achieve this, we used Toloka.ai, a crowdsourcing platform where a diverse pool of evaluators assessed the relevance of AI-generated illustrations to specific excerpts of fairy tales. This methodology allowed for a broad-based, democratized approach to understanding how illustrations created by AI are perceived in terms of accuracy, engagement, and emotional resonance with the source material. Our findings reveal that DALL-E 3 demonstrates a significant potential in creating visually appealing, contextually appropriate, and emotionally resonant illustrations that can rival traditional artistic methods, with a relevance rate of 62%. However, the conducted research shows that images generated by DALL-E 3 are often irrelevant and inconsistent, which makes it hard to use a system like this in a fully automated fashion without human supervision. This research contributes to the emerging field of AI in creative industries, offering insights for developers, educators, and storytellers seeking innovative approaches to enrich children's literary experiences.**

Keywords-DALL-E 3; children's fairy tales; automatic illustration generation; AI-generated illustrations; visual storytelling.

I. Introduction

In the realm of children's literature, illustrations play a pivotal role in bringing stories to life, offering visual narratives that complement and enhance the textual content. Traditionally, this task has been the domain of human artists, whose creativity and interpretation add depth and emotion to the tales. However, the advent of AI technologies, particularly in the field of generative art, presents novel opportunities and challenges for the creation of such illustrations.

The development of AI models like DALL-E 3 by OpenAI signifies a breakthrough in our ability to generate detailed, context-specific images from textual descriptions. This technology harbors the potential to revolutionize the way illustrations are produced for children's fairy tales, making it possible to automate the creation of artwork that resonates with the narratives and characters of these stories. The implications for storytelling, publishing, and educational content are profound, offering new avenues for engagement and creativity.

Despite the promise, the application of AI in creating illustrations for children's literature raises questions regarding the quality, relevance, and emotional connection of the generated images. Can an AI like DALL-E 3 understand and interpret fairy tales well enough to produce illustrations that capture the essence of these stories? How do these AI-generated illustrations compare to those created by human artists in terms of appealing to children's imaginations and understanding? Will they relate to the story or not?

To address these questions, our study investigates the capability of DALL-E 3 to generate illustrations for children's fairy tales from text descriptions. We used Toloka.ai, a crowdsourcing platform, to assess the relevance of the AI-generated illustrations.

This paper aims to contribute to the emerging discourse on the intersection of AI and creative arts, specifically within the context of children's literature. By exploring the potential of DALL-E 3 to create meaningful and engaging illustrations, we seek to understand the role that AI can play in enriching storytelling and whether it can serve as a viable tool for authors and publishers in the future.

In Section 2, the paper explores related works in the field of automatic generation of illustrations for children's fairy tales from text. Section 3 outlines the methodology adopted in the development of an AI-powered application designed to automatically generate illustrations for children's fairy tales. Section 4 details the implementation of the AI-powered application, including the technical architecture and the integration of the Generative Pre-trained Transformer 4 (GPT-4) and DALL-E 3. Section 5 presents the performace evaluation of the solution, highlighting the relevance of the images. Finally, Section 6 concludes the paper with a summary of the findings and implications for the future of AI in children's literature.

II. Related Works

The domain of automatic generation of illustrations for children's fairy tales from text represent a fascinating intersection of Natural Language Processing (NLP), computer vision, and creative AI technologies. This section reviews the related works that have contributed to the development of AI-powered applications, particularly focusing on technologies akin to DALL-E 3 for generating illustrations from textual content. The foundation for generating illustrations from text lies within the advancements of text-to-image synthesis. Reed et al. [1] were among the pioneers to explore this domain by conditioning deep convolutional Generative Adversarial Networks (GANs) on text features, enabling the generation of realistic images from descriptive text. Their work paved the way for subsequent innovations in image synthesis from textual descriptions. GANs have undergone significant evolution since their introduction by Goodfellow et al. [2]. The development of variants such as Conditional GANs (cGANs) and, later, Big GANs [3], has drastically improved the quality and resolution of generated images, making them more applicable for creating detailed illustrations. Parallel to the advancements in GANs, progress in language models, particularly transformer-based architectures like Bidirectional Encoder Representations from Transformers (BERT) [4] and GPT [5], has significantly improved the understanding and generation of natural language. DALL-E, introduced by OpenAI [6], marked a significant milestone in text-to-image generation by effectively combining the prowess of GPT-4 [7] with image synthesis capabilities. Its successor, DALL-E 2 [8], and the latest, DALL-E 3 [9], have further refined this approach, offering unprecedented quality and creativity in generating images from textual descriptions. These models have demonstrated a remarkable ability to understand complex descriptions and generate coherent and contextually relevant illustrations, making them ideal for creating children's book illustrations. Several studies have explored the application of AI in children's literature, particularly in generating illustrations. The work of Zakraoui et al. on "Visualizing Children Stories with Generated Image Sequences" [10] represents a significant contribution to the field, demonstrating the potential of using AI to create sequences of images that narrate children's stories. This approach underscores the evolving landscape of AI applications in children's literature focusing on the ability to visually narrate stories through a series of generated images, enriching the storytelling experience.

III. Method

This section outlines the methodology adopted in the development of an AI-powered application designed to automatically generate illustrations for children's fairy tales. The process leverages two cutting-edge AI technologies: GPT-4 for story generation and DALL-E 3 for the creation of corresponding illustrations. The integration of these technologies facilitates the end-to-end generation of illustrated fairy tales, from textual content creation to visual representation. The initial phase of the methodology involves generating the textual content of the fairy tales. For this purpose, OpenAI's GPT-4, an advanced iteration of the Generative Pre-trained Transformer models are utilized. GPT-4's ability to understand and generate human-like text makes it an ideal tool for crafting creative and engaging stories. The process is initiated by providing GPT-4 with a seed prompt, which outlines the desired theme or elements to be included in the fairy tale. The model then generates a story based on this prompt, ensuring originality and thematic relevance. The generated stories are structured to include clear narrative elements such as setting, characters, conflict, and resolution, making them suitable for children's literature. Following the creation of the textual content, DALL-E 3 is employed to generate illustrations corresponding to specific parts of the fairy tales. DALL-E 3, a state-of-the-art text-to-image generation model developed by OpenAI is capable of creating detailed and contextually relevant images from textual descriptions. This capability is leveraged to translate selected excerpts of the generated fairy tales into visual illustrations. The generated stories and corresponding illustrations are integrated into a cohesive fairy tale book format. This integration involves pairing each illustration with the relevant portion of the text, ensuring a logical and seamless narrative flow. Following the initial integration, a refinement process is undertaken. This process includes reviewing the coherence between the text and illustrations, the narrative flow, and the overall aesthetic appeal. Adjustments may involve regenerating certain illustrations with modified descriptions to better capture the intended scene or revising parts of the text for clarity or impact. The methodology described herein represents a novel approach to leveraging AI for the creative task of generating illustrated children's fairy tales. By combining the linguistic capabilities of GPT-4 with the visual creativity of DALL-E 3, this work contributes to the expanding field of AI-assisted content creation, opening new possibilities for storytelling and illustration.

IV. Implementation

The implementation of the AI-powered application for generating illustrated children's fairy tales involved the development of a frontend interface using TypeScript with React [11] and a backend server utilizing Node.js [12]. This section details the technical architecture, the integration of the GPT-4 and DALL-E 3 APIs for content and illustration generation and the overall workflow of the application. The application architecture is divided into two main components: the frontend and the backend. The frontend serves as the user interface, allowing users to input initial prompts for story generation and to interact with the generated content. It is developed using TypeScript, a statically typed superset of JavaScript, which provides enhanced development experience and maintainability through strong typing and object-oriented features. The frontend communicates with the backend server via RESTful APIs to request story generation and illustration services. The core functionality of the application relies on the integration with OpenAI's GPT-4 and DALL-E 3 APIs. The process begins with the backend server receiving a prompt from the frontend, which it forwards to the GPT-4 API to generate the fairy tale text. The code of the feature creating fairy tales and illustrations is shown in Figure 1. The GPT-4 API is called with a specific prompt to control the length, style, and thematic elements of the generated story, ensuring it aligns with the user's input and is suitable for a children's fairy tale. We ask the GPT-4 to return the response in JavaScript Object Notation (JSON) format. The decision to utilize JSON objects for structuring the fairy tale text was driven by the need for a flexible and easily parseable format that could accommodate the dynamic nature of story generation and illustration.

For creating a fairy tale, we use the following prompt:

`const NEWPROMPT` = Generate a fairy tale in the genre of `SELECTEDGENRE` featuring a `CHARACTERGENDER` protagonist named `CHARACTERNAME`. Generate a response in JSON format!!! Each paragraph of the fairy tale should be encapsulated as a value within a distinct key-value pair, where keys are sequential numbers starting from 1 and it is always a number and value is always a string. The tale should

```
const COMPLETION = await openai.chat.
    completions.create({
  messages: [{ role: "system",
      content: NEWPROMPT }],
  model: "gpt-4-1106-preview",
  response_format: { type: "
      json_object" },
});
const STORY = JSON.parse(completion?.
    choices[0]?.message.content)
```

Figure 1. Creating a fairy tale

incorporate `ADDITIONALPROMPT` to enrich the storyline. Ensure that each value contains a mini-narrative of 3 to 5 sentences, facilitating their use as prompts for image generation. Structure the JSON object so that it is easily parseable, with each paragraph kept under 2000 characters and segmented into individual sentences as separate key-value pairs. This format will support direct use of the sentences as prompts for creating corresponding images. Please maintain a coherent and engaging narrative flow throughout the fairy tale, adhering to the specified genre and character details. Do not include any explicit content, and ensure that the tale is suitable for children. Do not use words that are flagged by OpenAI content policies. Use words up that a 12 year old would understand.

The design of the prompts was guided by the need to generate fairy tales that are imaginative, culturally sensitive, and age-appropriate. The prompt structure was carefully crafted to ensure the generated stories and illustrations would captivate children's interest while adhering to ethical and content standards.

The JavaScript Object Notation (JSON) object is structured to facilitate the extraction of individual paragraphs, each of which serves as a prompt. The generated fairy tale is embeded in the following prompt which was developed with the intention to create visuals that complement the story's narrative, emphasizing a vibrant, engaging, and wholesome environment:

`const ILLUSTRATIONPROMPT` = Create a vibrant and engaging illustration suitable for a children's book, capturing the essence of an adventurous story. The scene should vivisd `STORY`, bringing to life the protagonist journey with rich, colorful imagery that appeals to young readers. The illustration should be wholesome, filled with elements of nature such as trees, flowers, and perhaps friendly forest animals, to complement the story's theme. It is crucial that the image is free from any form of violence, explicit content, or text, ensuring it is appropriate for a children's audience and adheres to publishing standards for educational and entertaining material. The art style should be warm and inviting, with a focus on creating an immersive experience for children to imagine themselves alongside the protagonist on its journey.

The directive to include elements of nature and ensure the absence of violence or explicit content in the prompt reflects our commitment to fostering a safe and positive reading environment for children. The choice of a warm and inviting art style was made to encourage children's imagination and identification with the story's protagonist and their adventures.

```
openai.images.generate({
  model: "dall-e-3",
  ILLUSTRATIONPROMPT,
  size: "1024x1024",
  style: 'natural',
}).then(image => image?.data[0]?.url).
    catch(error => {
  console.error(`Error generating image
      for prompt ${i}:`, error);
  return null;
});
```

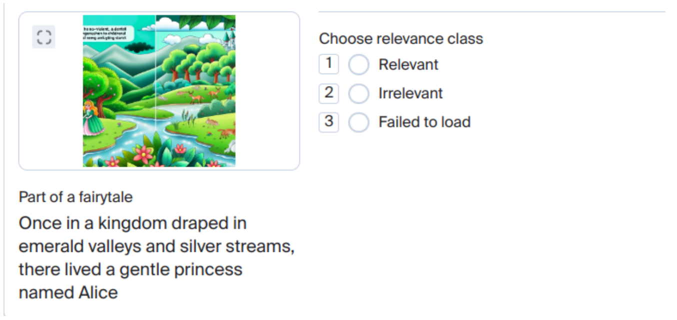
Figure 2. Function for creating a fairy tale and generating illustrations

Then, we create illustrations using the following code in Figure 2.

Once the story is generated, the backend server identifies key scenes or elements within the text that would benefit from illustration. For each of these, a descriptive text snippet is extracted or formulated and sent to the DALL-E 3 API to generate corresponding illustrations. The DALL-E 3 API's capability to create high-quality images from textual descriptions allows for the generation of visually appealing and contextually relevant illustrations for the fairy tale. The user initiates the story creation process via the frontend interface by providing a seed prompt or theme for the fairy tale. The frontend then communicates with the backend server, which orchestrates the calls to the GPT-4 and DALL-E 3 APIs. The generated story and illustrations are returned to the frontend, where they are displayed to the user in an interactive, book-like format. Users have the option to regenerate specific parts of the story or illustrations if they desire alterations. The frontend interface provides tools for users to select parts of the story for regeneration and to submit new prompts or descriptions for illustrations, enhancing the interactive and customizable nature of the application.

V. Performance

To evaluate the performance and relevance of the AI-generated illustrations to the specific excerpts of fairy tales, an extensive evaluation was conducted using Toloka.ai, a crowd-sourcing platform known for its diverse pool of evaluators. The primary objective of the evaluation was to determine the extent to which the AI-generated illustrations accurately reflected the content and spirit of the fairy tale excerpts they were associated with. To achieve this, a task framework was designed for evaluators on Toloka.ai, guiding them through a systematic assessment process.

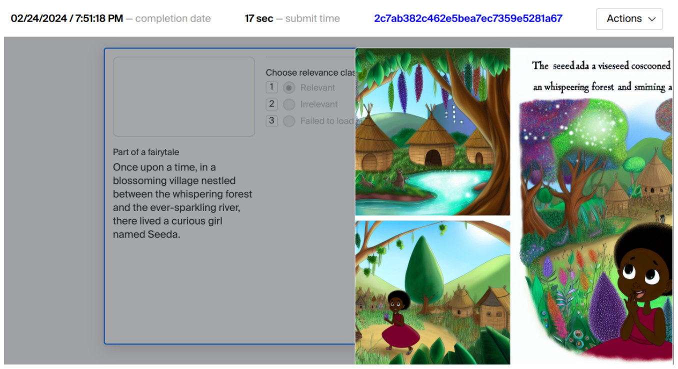

Choose relevance class

1 ○ Relevant
2 ○ Irrelevant
3 ○ Failed to load

Part of a fairytale

Once in a kingdom draped in emerald valleys and silver streams, there lived a gentle princess named Alice

Figure 3. Evaluation sample


02/24/2024 / 7:51:18 PM – completion date 17 sec – submit time 2c7ab382c462e5bea7ec7359e5281a67

Part of a fairytale

Once upon a time, in a blossoming village nestled between the whispering forest and the ever-sparkling river, there lived a curious girl named Seeda.

Figure 4. Response sample 1

Evaluators were presented with an AI-generated illustration alongside a specific excerpt from a fairy tale as in Figure 3. Their task was to judge the relevance of the illustration to the text, considering whether the visual elements accurately represented the narrative details and mood of the story segment. The task instructions provided to evaluators were as follows:

“Examine the illustration and Title: Evaluators were instructed to start by observing the illustration and reading its accompanying title, both of which were intended to encapsulate a segment of a fairy tale.

Make your decision: Based on their examination, evaluators were asked to classify the illustration as either “Relevant” or “Irrelevant” to the part of the fairy tale it was supposed to depict.

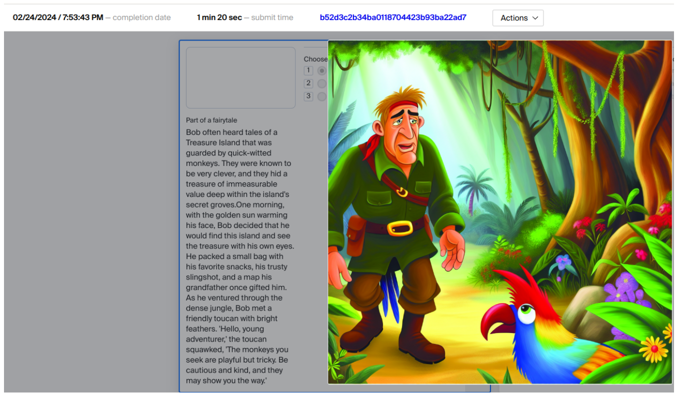
Adjust image size if necessary: An option to enlarge the illustration was available to ensure evaluators could closely inspect the image for detailed assessment.”

The criteria for relevance were clearly defined:

“Relevant: The illustration was considered relevant if it directly corresponded with the narrative details of the fairy tale segment, meaning all key elements mentioned or implied in the text were visually represented.

Irrelevant: An illustration was marked as irrelevant if it significantly diverged from the fairy tale segment by depicting unrelated elements, omitting critical details, or if the title and content of the illustration did not align with the narrative segment under evaluation.”

Figure 4 and Figure 5 are representing examples of the


02/24/2024 / 7:53:43 PM – completion date 1 min 20 sec – submit time b52d3c2b34ba0118704423b93ba22ad7

Part of a fairytale

Bob often heard tales of a Treasure Island that was guarded by quick-witted monkeys. They were known to be very clever, and they hid a treasure of immeasurable value deep within the island's secret groves.One morning, with the golden sun warming his face, Bob decided that he would find this island and see the treasure with his own eyes. He packed a small bag with his favorite snacks, his trusty slingshot, and a map his grandfather once gifted him. As he ventured through the dense jungle, Bob met a friendly toucan with bright feathers. 'Hello, young adventurer,' the toucan squawked, 'The monkeys you seek are playful but tricky. Be cautious and kind, and they may show you the way.'

Figure 5. Response sample 2

evaluation’s results. Involving 134 evaluators, the study covered 473 distinct evaluations, offering a broad perspective on the effectiveness of AI in producing contextually appropriate visuals. According to the collected data, 295 evaluations deemed the illustrations relevant, indicating a successful match between the illustration and the narrative content of the fairy tale excerpt. Conversely, 178 evaluations found the illustrations irrelevant, pointing to a discrepancy between the visual representation and the story segment.

These findings offer a nuanced view of the AI’s performance, showcasing a substantial success rate in generating illustrations that align well with the fairy tales’ narrative elements. However, the presence of a significant number of illustrations judged as irrelevant also highlights areas where the AI’s understanding and interpretation of text can be further improved. The insights gained from this extensive feedback loop are invaluable for future enhancements of AI models, aiming to increase the precision and relevance of generated illustrations in subsequent fairy tale projects. This evaluation not only demonstrates the AI’s potential in creating visually engaging content for children’s literature but also emphasizes the importance of continuous refinement in AI applications to better serve creative storytelling.

VI. Conclusion

This paper presented an innovative approach to enriching children’s fairy tales with AI-generated illustrations, leveraging the capabilities of GPT-4 for text generation and DALL-E 3 for visual content creation. Through the integration of advanced AI technologies, we have developed an application that automates the process of creating illustrated fairy tales, offering a novel interactive experience for both children and adults alike.

Our methodology involved using the GPT-4 API to generate fairy tale narratives based on user prompts and employing the DALL-E 3 API to produce illustrations that visually complement specific excerpts from these tales. The implementation showcased the seamless collaboration between frontend technologies like TypeScript and backend services powered by Node.js, culminating in a user-friendly platform that bridges the gap between traditional storytelling and modern technological innovations. However, there are still possibilities for further improvements. Future plans include:

- to explore innovative methods that allow the selection of the text excerpts to create illustrations autonomously;
- to incorporate additional quality metrics into our evaluation process, including visual appeal and emotional resonance;
- to compare existing solutions such as Midjourney and DreamStudio for generating illustrated children's fairy tales.

The performance assessment carried out through the crowdsourcing platform Toloka.ai, yielded significant observations regarding the relevance and precision of the AI-generated illustrations. With a relevance rate of 62.3%, as indicated by 295 out of 473 evaluations affirming the alignment between illustrations and their corresponding fairy tale excerpts, these findings illuminate the capability of AI in generating visually engaging and contextually fitting illustrations for literary compositions. However, this evaluation also spotlighted areas necessitating enhancement, particularly the need to improve the AI's interpretation of complex narrative nuances aimed at diminishing the occurrence of illustrations deemed irrelevant.

In conclusion, this project demonstrates the feasibility of using AI to automate the generation of illustrated children's fairy tales. The findings from this study underscore the importance of continuous refinement in AI model training and the potential for future research to further enhance the storytelling experience. By pushing the boundaries of what is possible with AI, we can provide enriching and immersive literary experiences that captivate the imagination of readers of all ages.

References

[1] S. Reed *et al.*, "Generative adversarial text to image synthesis," in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., ser. Proceedings of Machine Learning Research, vol. 48, New York, New York, USA: PMLR, 2016, pp. 1060–1069. [Online]. Available: https://proceedings.mlr.press/v48/reed16.html [retrieved: Mar. 2024].

[2] I. J. Goodfellow *et al.*, "Generative adversarial nets," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, MIT Press, 2014, pp. 2672–2680. [retrieved: Mar. 2024].

[3] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018. [retrieved: Feb. 2024].

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018. [retrieved: Feb. 2024].

[5] A. Radford *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019. [retrieved: Feb. 2024].

[6] M. D. M. Reddy, M. S. M. Basha, M. M. C. Hari, and M. N. Penchalaiah, "Dall-e: Creating images from text," *UGC Care Group I Journal*, vol. 8, no. 14, pp. 71–75, 2021.

[7] J. Achiam *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023. [retrieved: Feb. 2024].

[8] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv preprint arXiv:2204.06125*, vol. 1, no. 2, p. 3, 2022.

[9] J. Betker *et al.*, "Improving image generation with better captions," *Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf*, vol. 2, no. 3, p. 8, 2023.

[10] J. Zakraoui, M. Saleh, S. Al-Maadeed, J. M. Alja'am, and M. S. Abou El-Seoud, "Visualizing children stories with generated image sequences," in *Visions and Concepts for Education 4.0: Proceedings of the 9th International Conference on Interactive Collaborative and Blended Learning (ICBL2020)*, Springer, 2021, pp. 512–519.

[11] Jordan Walke, *React*, version 18, 2022. [Online]. Available: https://react.dev [retrieved: Mar. 2024].

[12] Ryan Dahl, *Node.js*, version 18.19.0, 2023. [Online]. Available: https://nodejs.org [retrieved: Mar. 2024].