

An Empirical Investigation into the Use of Image Captioning for Automated Software Documentation

Kevin Moran,
Ali Yachnes,
George Purnell,
Junayed Mahmud,
Michele Tufano,
Carlos Bernal-Cardenas
Denys Poshyvanyk
Zach H'Doubler

29th IEEE International Conference on Software Analysis,
Evolution and Reengineering

Wednesday, March 16th, 2022

THE IMPORTANCE OF SOFTWARE DOCUMENTATION

```
function check_things($first_name, $last_name, $age) {  
    if (  
        !ctype_alpha($_POST['first_name']) OR  
        !ctype_alpha($_POST['last_name']) OR  
        !ctype_digit($_POST['age'])  
    ) {  
        return false;  
    }  
    return true;  
}
```

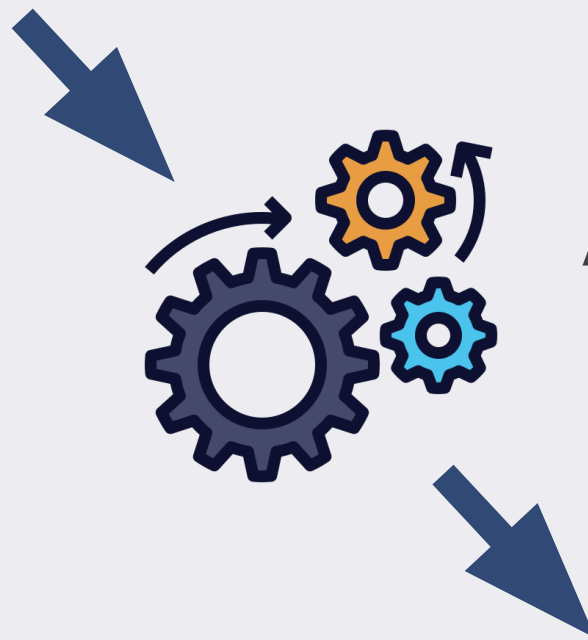
THE IMPORTANCE OF SOFTWARE DOCUMENTATION

// Check if the user input for the first name, last name
and age match alphabetic and numeric characters respectively

```
function check_things($first_name, $last_name, $age) {  
    if (  
        !ctype_alpha($_POST['first_name']) OR  
        !ctype_alpha($_POST['last_name']) OR  
        !ctype_digit($_POST['age'])  
    ) {  
        return false;  
    }  
    return true;  
}
```

AUTOMATED SOFTWARE DOCUMENTATION

```
function check_things($first_name, $last_name, $age) {  
    if (  
        !ctype_alpha($_POST['first_name']) OR  
        !ctype_alpha($_POST['last_name']) OR  
        !ctype_digit($_POST['age'])  
    ) {  
        return false;  
    }  
    return true;  
}
```



Automated Software
Documentation Tool

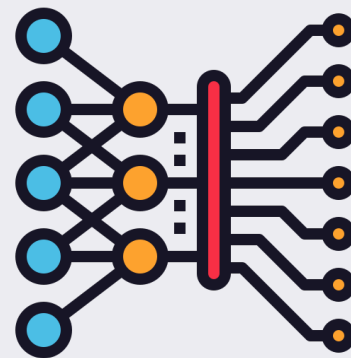
// Check if the user input for the first name, last name
and age match alphabetic and numeric characters respectively

```
function check_things($first_name, $last_name, $age) {  
    if (  
        !ctype_alpha($_POST['first_name']) OR  
        !ctype_alpha($_POST['last_name']) OR  
        !ctype_digit($_POST['age'])  
    ) {  
        return false;  
    }  
    return true;  
}
```

4 }

SOFTWARE DOCUMENTATION AS MACHINE TRANSLATION

```
function check_things($first_name, $last_name, $age) {  
  if (  
    !ctype_alpha($_POST['first_name']) OR  
    !ctype_alpha($_POST['last_name']) OR  
    !ctype_digit($_POST['age'])  
  ) {  
    return false;  
  }  
  return true;  
}
```



Neural Machine
Translation Model



```
// Check if the user input for the first name,  
last name  
and age match alphabetic and numeric  
characters respectively
```

Automatically Generating Commit Messages

Summarization of Source Code Changes

Luis Fernando Cortés-Coy¹, Mario Linares-Vásquez², Jairo Aponte¹, Denys Poshyvanyk²
¹ Universidad Nacional de Colombia, Bogotá, Colombia
² The College of William and Mary, Williamsburg, VA, USA
lfcortesco@unal.edu.co, mlinarev@cs.wm.edu, jhapontem@unal.edu.co, denys@cs.wm.edu

Abstract—Although version control systems allow developers to be and explain the rationale behind code changes in messages, the state of practice indicates that most of the commit messages are either very short or even empty. In a study of 23K+ Java projects it has been found that the messages are descriptive and over 66% of those contain fewer words as compared to a typical English sentence (5-20 words). However, accurate and complete summarizing software changes are important to support development and maintenance tasks. In this paper, we propose an approach, coined as *ChangeScribe*, which automatically generates natural language commit messages. *ChangeScribe* generates automatically from the commit message stereotype, the type of changes done only to property files), and the underlying changes. We evaluated the approach involving 23 developers in which the automatically generated commit messages were compared them with commit messages generated by developers of six open source projects. The results show that the automatically generated commit messages are about 66% as descriptive as the manually generated ones. Only 10% of analyzed messages were descriptive.

One possible explanation behind this dissonance between theory and practice has been recently explored in several studies [26], [4], [20]. In particular, it has been observed that the number and nature of daily activities by developers, including a large number of interactions, influence their attention to modified code. These daily activities become one of the most important or forgetting activities become one of the most important commit time [20]. Moreover, the exact set of changes and expensive activities are multiple.

One possible explanation behind this discrepancy and only theory and practice has been recently studies [26], [4], [20]. In particular, the number and nature of authors, including nature of their affiliations, may be a significant factor.

English sentence (i.e., 15 - 20 words) and analyzed messages were descriptive.

One possible explanation behind this dissonance between theory and practice has been recently explored in several studies [26], [4], [20]. In particular, it has been observed that the number and nature of daily activities by developers, including a large number of interruptions influence their attention to modified code or forgetting activities become one of the main reasons for committing errors. More specifically, the exact set of changes and expensive operations are often forgotten during multiple editing sessions.

**Encoding for
Summarization of Sub**

Aakash Bansal, Sakib Haque, and Collin McMillan
Dept. of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
{abansal1, shaque, cmc}@nd.edu

the roots in machine tr
a vectorized rep
English),
se

Abstract—Source code summarization of a subroutine is the task of writing a short, natural language description of that subroutine. The description usually serves in documentation aimed at programmers, where even brief phrase (e.g. “compresses data to a zip file”) can help readers rapidly comprehend what a subroutine does on neural networks (and encoder-decoder model designs in particular) have established themselves as the state-of-the-art. Problem information needed to create a summary is present in the question of encoding source code summarization. Thus a current question of encoding source code summarization. In this paper, we propose models of code summarization. We create a vectorized state-of-the-art project, and show our

AUTOMATED SOFTWARE DOCUMENTATION

Action Word Prediction for Neural Source Code Summarization

Sakib Haque*, Aakash Bansal*, Lingfei Wu† and Collin McMillan*

*Dept. of Computer Science, University of Notre Dame, Notre Dame, IN, USA

{shaque, abansal, cmc}@nd.edu

†IBM Research, Yorktown Heights, NY, USA

{wuli}@us.ibm.edu

Abstract—Source code summarization is the task of creating short, natural language descriptions of source code. Code summarization is the backbone of much software documentation such as JavaDocs, in which very brief comments such as “adds the customer object” help programmers quickly understand a snippet of code. In recent years, automatic code summarization has become a high value target of research, with approaches based on neural networks making rapid progress. However, as we will show in this paper, the production of good summaries relies on the production of the action word in those summaries: the meaning of the example above would be completely changed if “removes” were substituted for “adds.” In this paper, we advocate for a special emphasis on action word prediction as an important stepping stone problem towards better code summarization – current techniques try to predict the action word along with the whole summary, and yet action word prediction on its own is quite difficult. We show the value of the problem for code summaries, explore the performance of current baselines, and provide recommendations for future research.

Index Terms—neural networks, source code summarization, automatic documentation generation, AI in SE

I. INTRODUCTION

The task of creating short, natural language descriptions of source code has come to be known as “source code summarization.” Code summarization is the backbone of a plethora of documentation such as JavaDocs [1], in which the natural language description (the “summary”) provides a quick way for programmers to understand the software’s components. Very often, these summaries are written for subroutines, so that programmers can read that a subroutine e.g. “computes the dot product of two vectors” rather than interpret the source code itself. Traditionally, programmers write these summaries around the time they write the code, to help other programmers in understanding that code.

Yet, as we will show in this paper, very often these techniques owe their good performance on their ability to predict the *first word* of the summary. Some of the reasons for this are technical: Existing techniques tend to be based on an encoder-decoder architecture (e.g. seq2seq, graph2seq) in which the output summary is predicted one word at a time. The first word is predicted first, then that first prediction is used to predict the second word, and so on. If the first word is wrong, the model can have a hard time recovering. This situation can be exaggerated by the aggressive use of attention mechanisms (as in Transformer-based models [3]), which can attend previous words in the predicted summary to parts of the source code. Often each subsequent word depends more and more on the previous predictions.

A more fundamental reason the first word is important is that the first word tends to be the action word in code summaries. As we will show (and in line with style guides [1], [4]), summaries usually fall into a pattern where the action word not only occurs first, but sets the tone for the whole summary. Consider examples such as “initializes the microphone for the web conference”, “sets the current speaker’s volume”, and “sorts the list of connected users.” A lot of information is communicated just by knowing that the code initializes, sets, or sorts. The rest of the summary depends on that information, begging the question: initializes/sets/sorts *what*?

The importance of early predictions in text generation models has been recognized in the NLP community for years, with several proposed technical workarounds e.g. beam search and alternative training strategies. Meanwhile, the prevalence of verb-direct object patterns in code summaries has long been observed in SE literature [5]. What is not yet recognized is the special importance of the action word in source code summarization, and how to leverage this importance to create

Neural Source Code Summarization of Subroutines

Aakash Bansal, and Collin McMillan

Department of Computer Science and Engineering

University of Notre Dame

Notre Dame, IN, USA

{abansal, cmc}@nd.edu

source code summary. Almost all neural approaches to code summarization are some form of an attentional encoder-decoder model, in which the encoder creates a vectorized representation of source code, and the decoder represents the natural language summary. In the past five years, neural approaches have almost completely superseded alternatives such as sentence templates or IR-based keyword extraction [8]–[10]. Current strategies to neural code summarization can be broadly classified by the type of information they focus on modeling in the encoder. There are approaches that treat code as text, focusing on the identifier names and other natural language content buried in code [11], [12]. There are approaches that encode the context of other code in the same source file [13]. Some techniques model the structure of source code with an RNN by linearizing structural representations such as the AST [12], [14], [15]. And there is very active study of GNN-based encoders to model structures such as the AST or CPG [16]–[18]. All of these lines of inquiry are showing promise and continue to advance the state of the art. Recent work in source code summarization has been focusing on providing models with ever more complex representations of code with the assumption that it will yield better and later predictions of code summaries [13], [15], [19], [20], [21]. This approach mirrors progress in many other fields such as machine translation or image captioning [21], [22]. However, hints from prior work point to a complementary relationship among neural models of code summarization, rather

arXiv:2101.02742v1 [cs.SE] 7 Jan 2021

SKEPTICISM OF NEURAL MACHINE TRANSLATION

Code to Comment “Translation”: Data, Metrics, Baseline & Evaluation

David Gros*, Hariharan Sezhiyan*, Prem Devanbu, Zhou Yu
University of California, Davis
{dgros,hsezhiyan,devanbu,joyu}@ucdavis.edu

ABSTRACT

The relationship of comments to code, and in particular, the task of generating useful comments given the code, has long been of interest. The earliest approaches have been based on strong syntactic theories of comment-structures, and relied on textual templates. More recently, researchers have applied deep-learning methods to this task—specifically, trainable generative translation models which are known to work very well for Natural Language translation (e.g., from German to English). We carefully examine the underlying assumption here: that the task of generating comments sufficiently resembles the task of translating between natural languages, and so similar models and evaluation metrics could be used. We analyze several recent code-comment datasets for this task: CODENN, DEEPCOM, FUNCOM, and DOCSTRING. We compare them with WMT19, a standard dataset frequently used to train state-of-the-art natural language translators. We found some interesting differences between the code-comment data and the WMT19 natural language data. Next, we describe and conduct some studies to calibrate BLEU (which is commonly used as a measure of comment quality), using “affinity pairs” of methods, from different projects, in the same project, in the same class, etc. Our study suggests that the current performance on some datasets might need to be improved substantially. We also argue that fairly naive information retrieval (IR) methods do well enough at this task to be considered a reasonable baseline. Finally, we make some suggestions on how our findings might be used in future research in this area.

ACM Reference Format:

David Gros*, Hariharan Sezhiyan*, Prem Devanbu, Zhou Yu. 2020. Code to Comment “Translation”: Data, Metrics, Baseline & Evaluation. In *Proceedings of ACM Conference (ASE '20)*. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnn>

1 INTRODUCTION

Programmers add comments to code to help comprehension. The value of these comments is well understood and accepted. A wide variety of comments exist [42] in code, including prefix comments (standardized in frameworks like Javadocs [31]) which are inserted before functions or methods or modules, to describe their function. Given the value of comments, and the effort required to write

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ASE '20, Sept 2020, Melbourne, Australia
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . \$15.00
<https://doi.org/10.1145/nnnnnnnn.nnnnnnn>

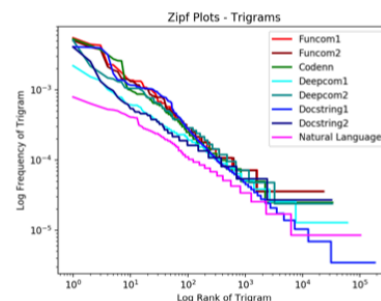


Figure 1: Distribution of trigrams in English (blue) in the WMT [10] German-English machine-translation dataset, and in English comments from several previously published Code-Comment datasets

them, there has been considerable interest in providing automated assistance to help developers to produce comments, and a variety of approaches have been proposed [38, 47, 48, 59].

Comments (especially prefix comments) are typically expected to be a useful summary of the function of the accompanying code. Comments could be viewed as a restatement of the semantics of the code, in a different and more accessible natural language; thus, it is possible to view comment generation as a kind of translation task, translating from one (programming) language to a another (natural) language. This view, together with the very large volumes of code (with accompanying comments) available in open-source projects, offers the very appealing possibility of leveraging decades of research in statistical natural language translation (NLT). If it's possible to learn to translate from one language to another from data, why not learn to synthesize comments from code? Several recent papers [22, 26, 33, 61] have explored the idea of applying Statistical Machine Translation (SMT) methods to *learn* to translate code to an English comments. But are these tasks really similar? We are interested to understand in more detail how similar the task of generating comments from code is to the task of translating between natural languages.

Comments form a domain-specific dialect, which is highly structured, with a lot of very repetitive templates. Comments often begin with patterns like “returns the”, “outputs the”, and “calculates the”. Indeed, most of the earlier work (which wasn't based on machine

“Natural Language NMT datasets show a stronger input-output dependence than Code to Comment translation datasets”

This suggests that code-to-comment translation is likely more difficult via neural machine translation due to an **information mismatch**

* Authors contributed equally

THE FUTURE OF AUTOMATED SOFTWARE DOCUMENTATION

On-Demand Developer Documentation

Martin P. Robillard*, Andrian Marcus†, Christoph Treude‡, Gabriele Bavota§, Oscar Chaparro†, Neil Ernst¶, Marco Aurélio Gerosa||, Michael Godfrey**, Michele Lanza§, Mario Linares-Vásquez††, Gail C. Murphy‡‡, Laura Morenoˆ, David Shepherdˆ, and Edmund Wong**

*McGill University, Canada

†The University of Texas at Dallas, USA

‡University of Adelaide, Australia

§Università della Svizzera italiana, Switzerland

¶University of Victoria, Canada

||Northern Arizona University, USA

**University of Waterloo, Canada

††Universidad de los Andes, Colombia

‡‡University of British Columbia, Canada

ˆColorado State University, USA

ˆABB, USA

Abstract—We advocate for a paradigm shift in supporting the information needs of developers, centered around the concept of *automated on-demand developer documentation*. Currently, developer information needs are fulfilled by asking experts or consulting documentation. Unfortunately, traditional documentation practices are inefficient because of, among others, the manual nature of its creation and the gap between the creators and consumers. We discuss the major challenges we face in realizing such a paradigm shift, highlight existing research that can be leveraged to this end, and promote opportunities for increased convergence in research on software documentation.

I. THE VISION

We advocate for a new vision for satisfying the information needs of developers, which we call On-Demand Developer Documentation (OD3). Development tasks typically involve a variety of artifacts, tools, processes, and other humans. Currently, when developers have questions, they may consult curated documentation, explore artifacts, browse Questions and Answers (Q&A) websites, or seek the advice of experts. Within this new perspective, an OD3 system would automatically generate high-quality documentation in response to a user query; the OD3 system would use a combination of knowledge extraction techniques on an underlying collection of structured and unstructured artifacts, including source code, issue tracking system metadata, and posts from Q&A forums. For example, a developer assigned to repair a fault related to copy-paste functionality might ask about the implementation of the system's Clipboard feature; in response, the OD3 system might generate a document that explains relevant design decisions for this feature (e.g., based on mining historical project data), and suggest alternatives (e.g., based on processing Q&A forum data). This paper is the outcome of a community effort; in the remainder, we motivate the need for OD3 and then discuss major research challenges that need to be addressed to realize a vision of OD3.

II. MOTIVATION

Documentation pervades many, if not most, software engineering activities [1], [2]. A particular type of documentation, which we call *developer documentation*, is specifically intended to assist software developers in the creation or modification of a system. Common types of developer documentation include source code comments, tutorials and reference documentation for application programming interfaces (APIs), and design documentation. Developer documentation is considered to be one of the most useful pieces of information by developers during software maintenance [1].

Although the ideal of fully self-documented software has been with us since the dawn of the discipline [3], the reality of software development technology and practice falls short. Documentation is an essential resource for creating and maintaining software systems, but it suffers from two fundamental limitations. First, it is costly to create and maintain, and second, it is a non-executable artifact whose presence and correctness are not technically critical to the construction of software. The combination of high cost and low immediate return on investment is particularly nefarious, and reports on documentation being a low priority task are routine [1], [4]. Over the years, tools have been developed to reduce some of the accidental inefficiencies related to the production of documentation. However, documentation tools provide relatively little help with the creation of original content.

Curated documentation can provide coherent and authoritative answers to some classes of questions, but the scope of such documentation is necessarily limited. The field has benefited from many studies of information needs of developers and maintainers [5], [6], [7], from which questions arise that would be hard to document, especially in the absence of a clear promise of return on investment.

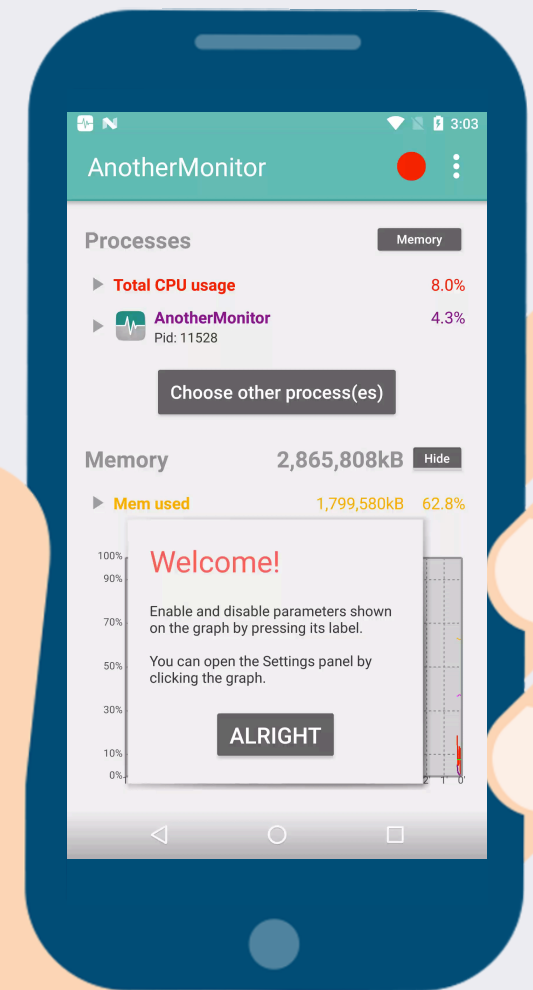
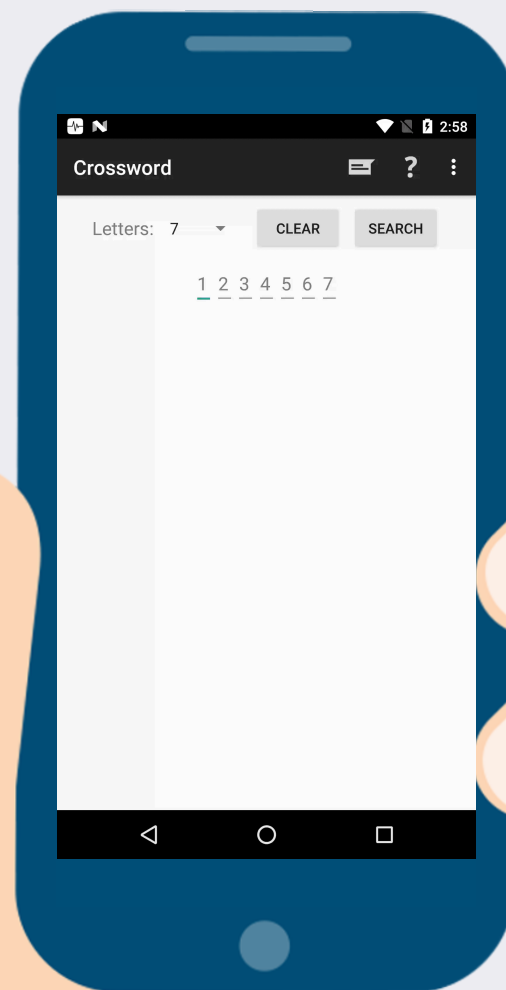
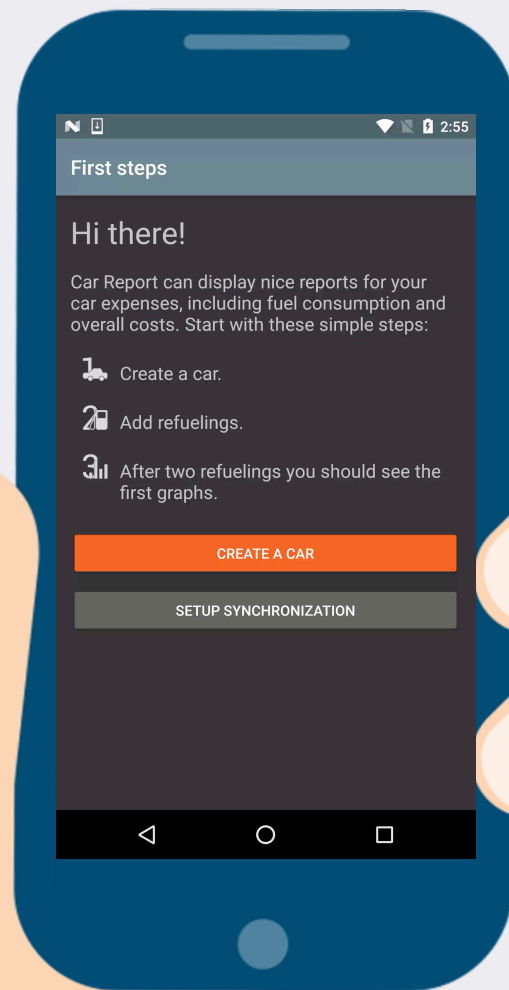
Research Challenges:

- Inferring undocumented program properties
- discovering latent abstractions and rationales

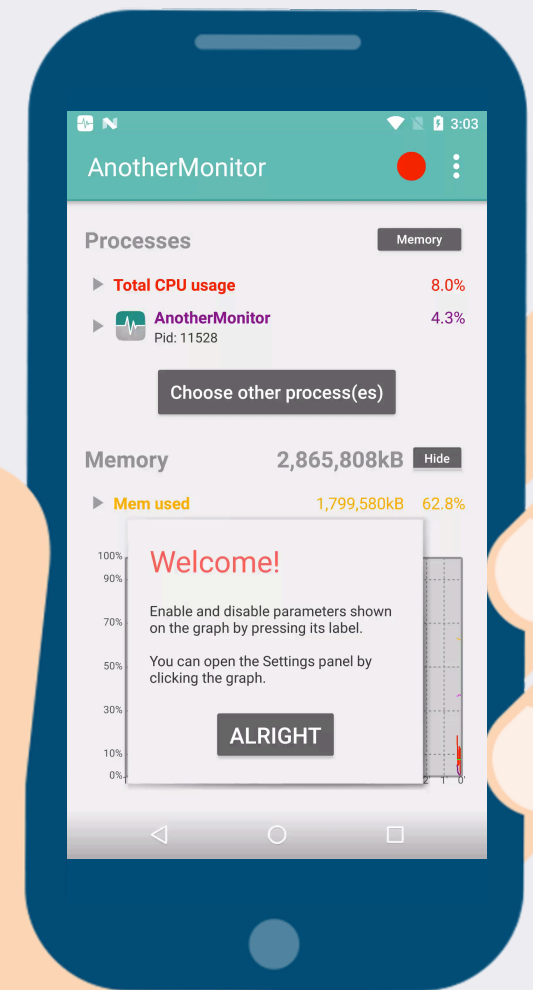
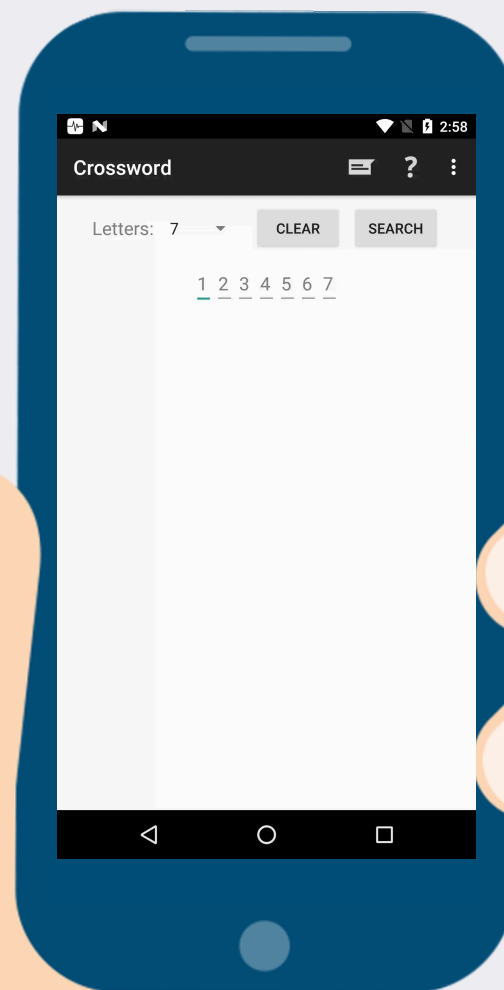
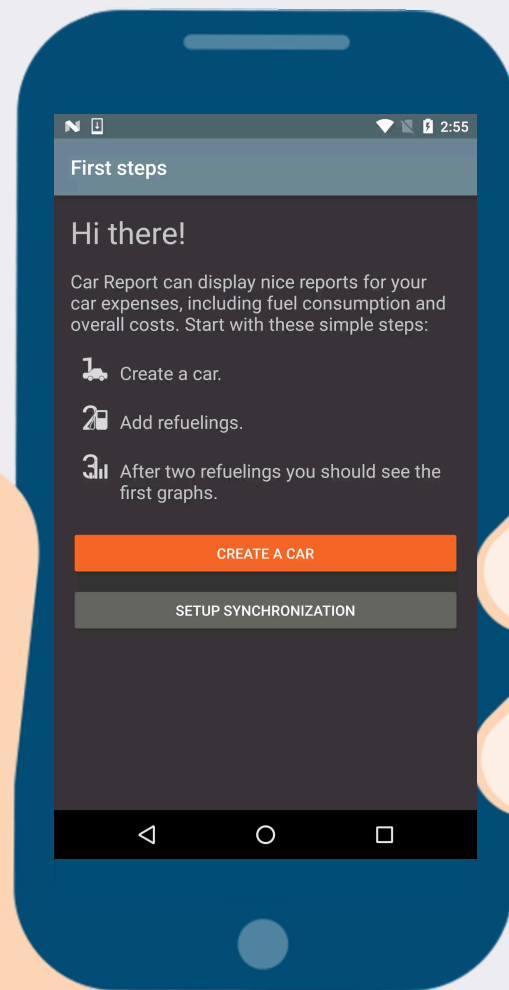
This suggests that we need to find *new techniques* and *data sources* to discover *hidden abstractions* of software to better automate documentation

KEY IDEA: INFERRING FUNCTIONALITY FROM UIs

KEY IDEA: INFERRING FUNCTIONALITY FROM UIs



KEY IDEA: INFERRING FUNCTIONALITY FROM UIs



NEURAL IMAGE CAPTIONING



"man in black shirt is playing guitar."



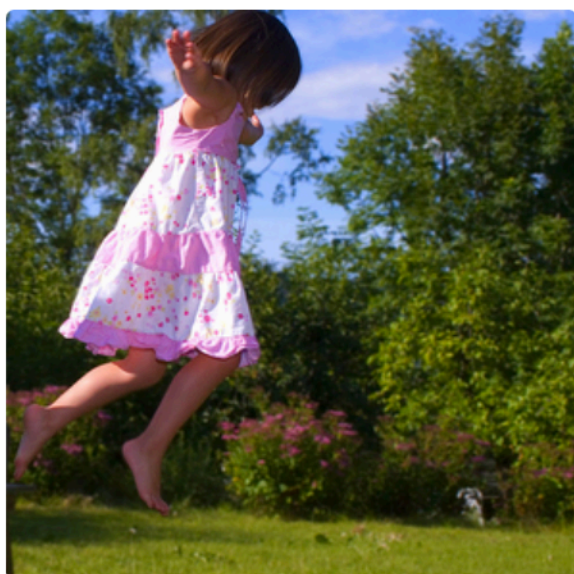
"construction worker in orange safety vest is working on road."



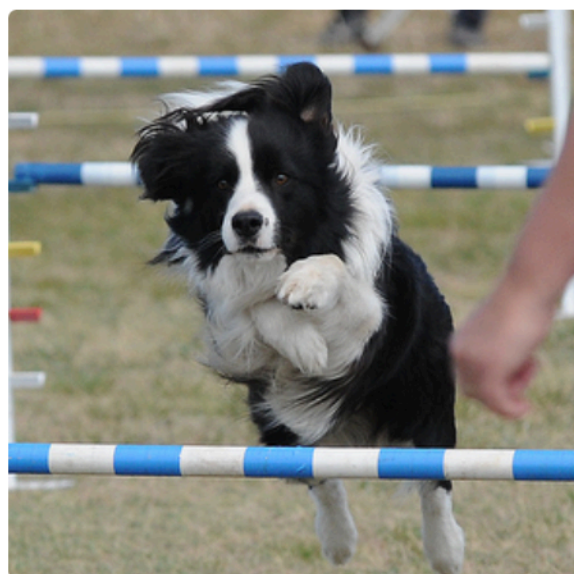
"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"girl in pink dress is jumping in air."



"black and white dog jumps over bar."

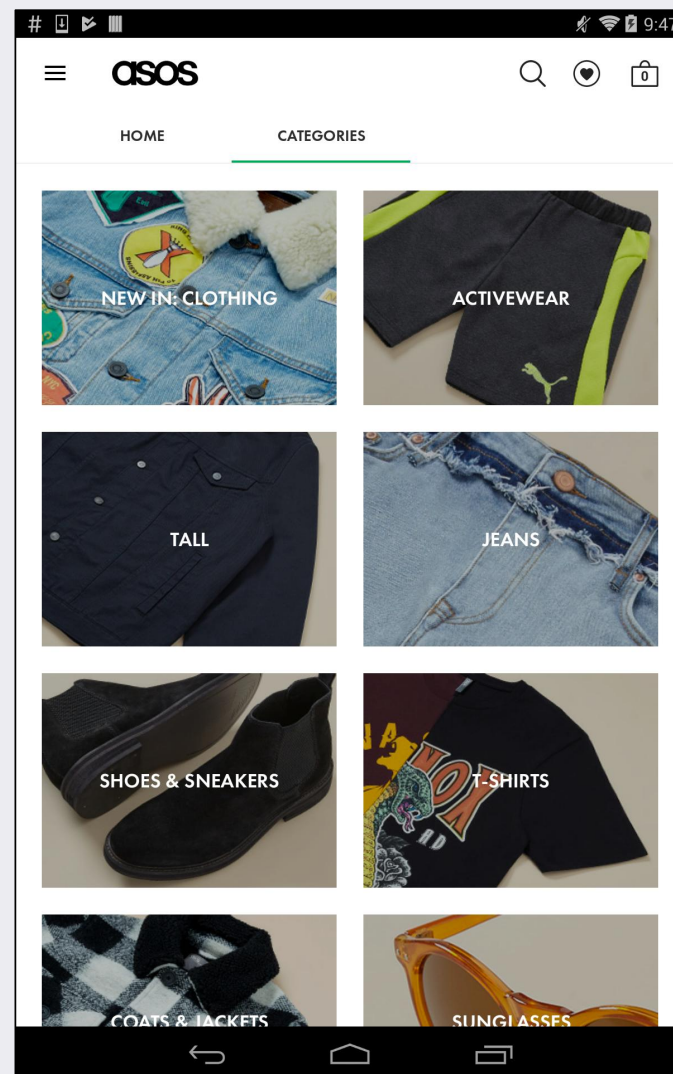


"young girl in pink shirt is swinging on swing."



"man in blue wetsuit is surfing on wave."

NEURAL IMAGE CAPTIONING OF UIs



High Level Caption

The screen allows the user to look at clothing categories

CLARITY PROJECT OVERVIEW

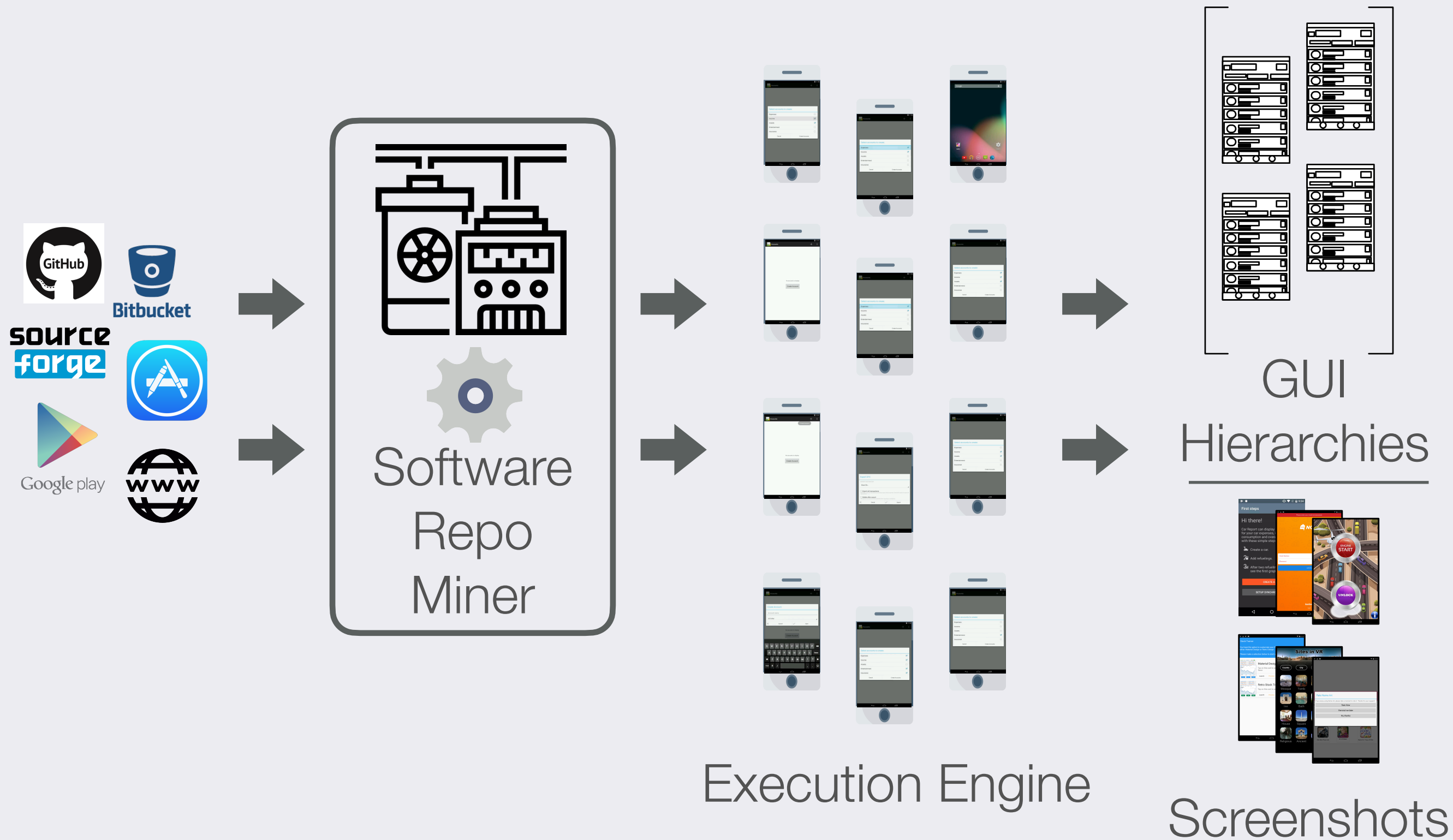
1 Clarity Dataset Collection
(Screenshots + GUI metadata + Captions)

2 Naturalness & Topic Analysis

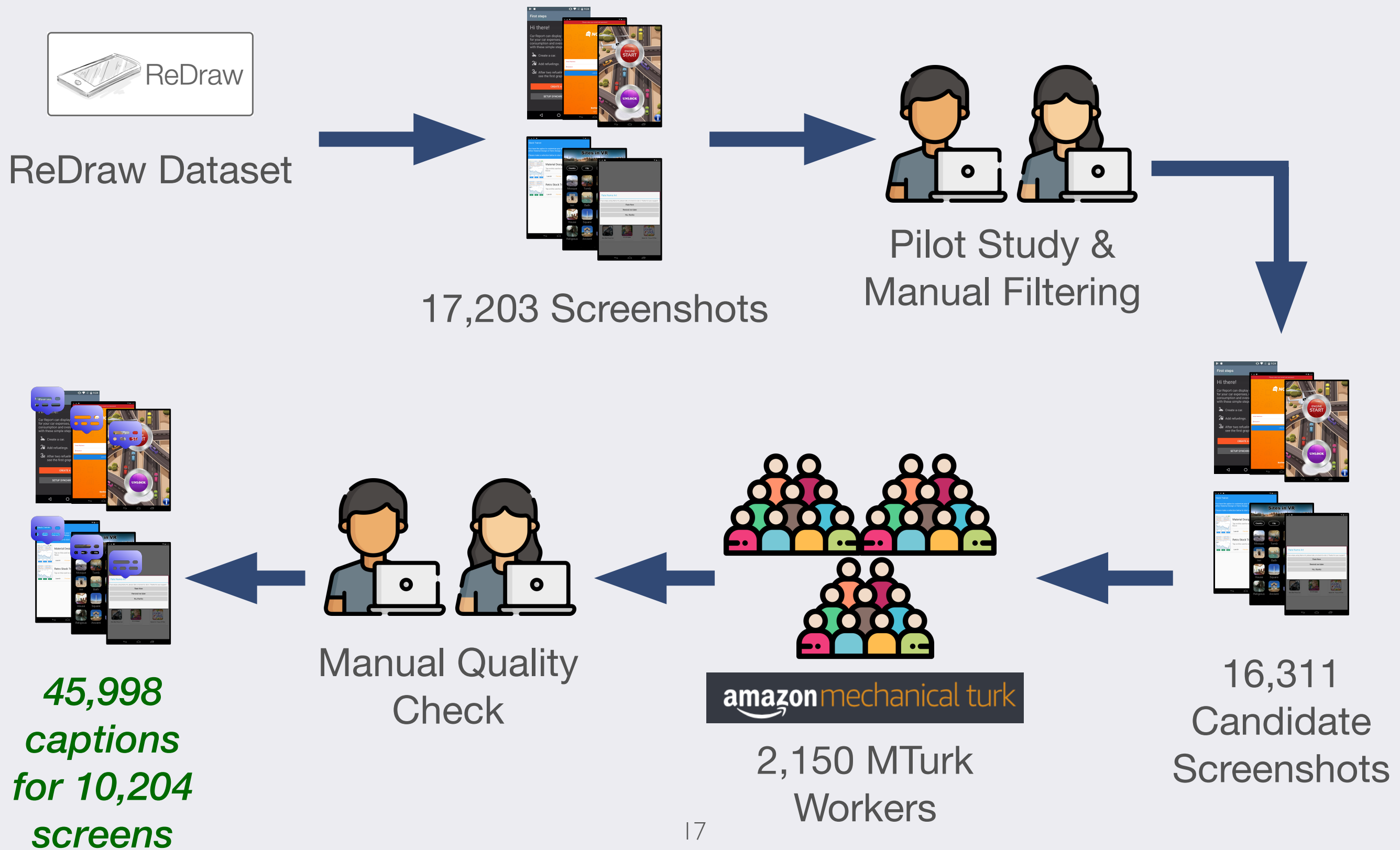
3 Train Image-Captioning and
Metadata Captioning Models

4 Quantitative and Qualitative
Model Evaluations

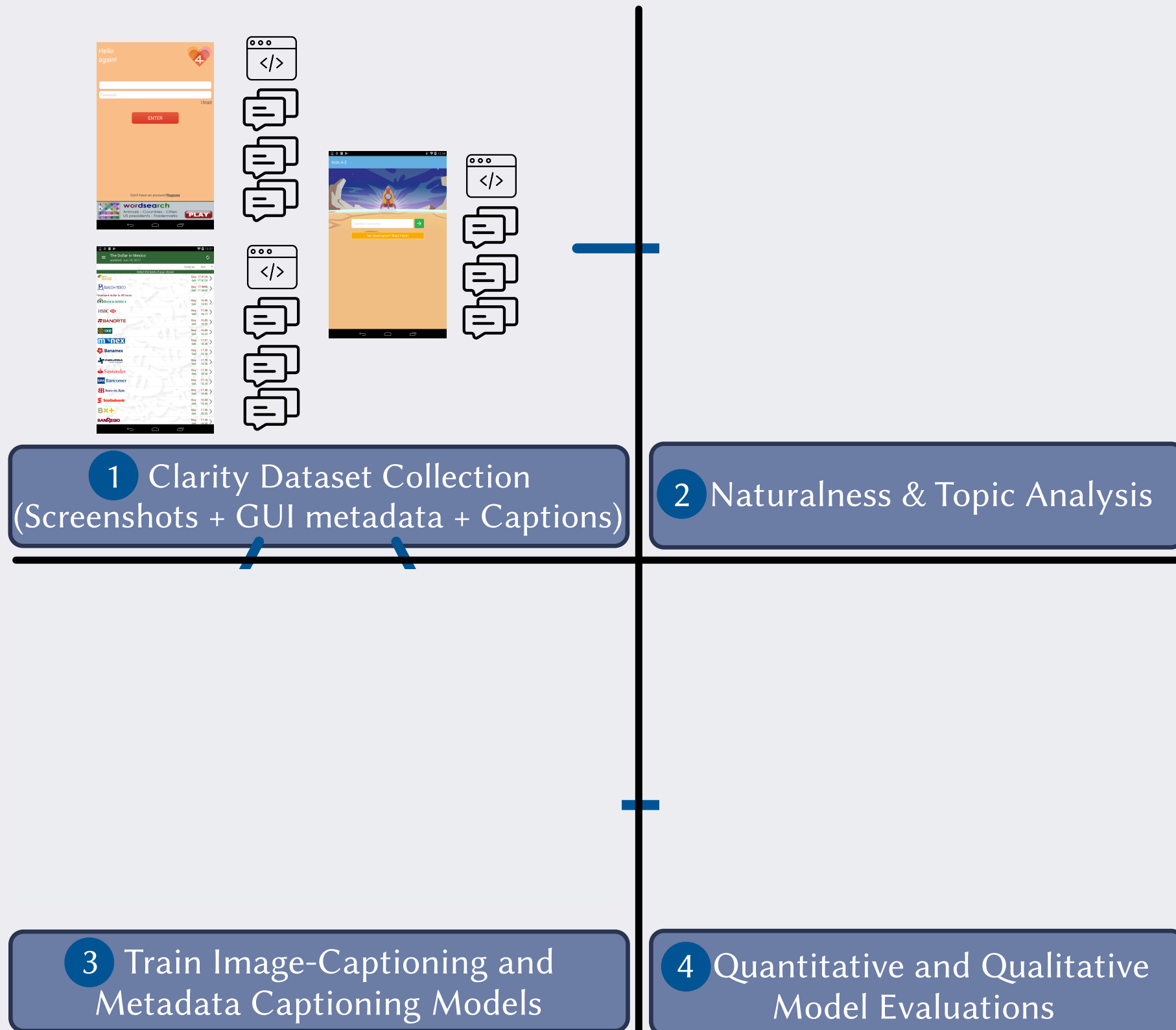
THE ReDRAW DATASET



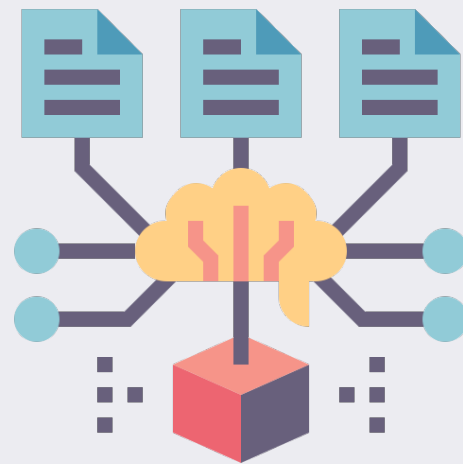
CLARITY DATASET COLLECTION



CLARITY PROJECT OVERVIEW



CROSS ENTROPY ANALYSIS

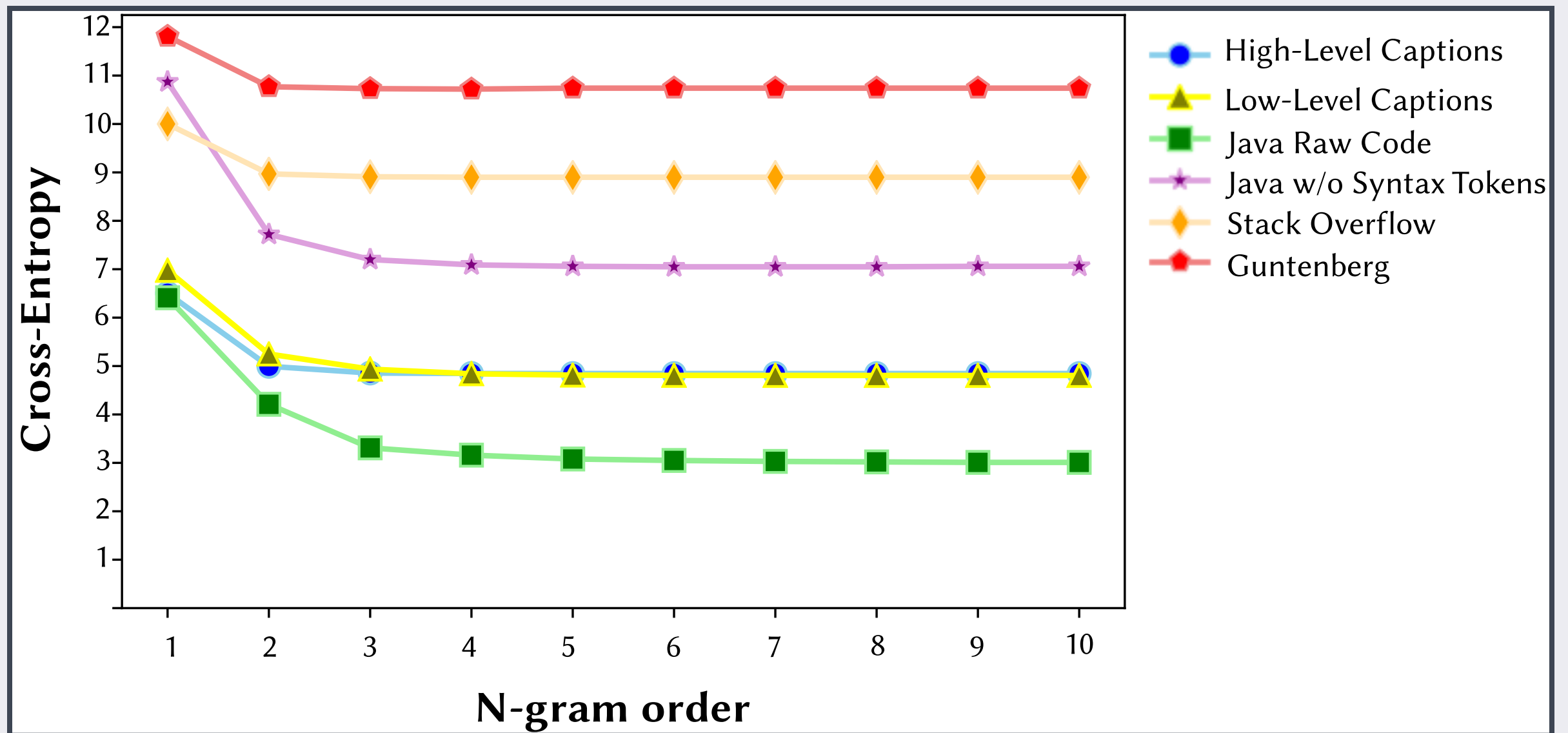


Clarity Captions
+
Code and Natural
Language Corpora

N-Gram
Language Model

Measure of
Cross Entropy

CROSS ENTROPY ANALYSIS - RESULTS



TOPIC MODELING ANALYSIS



Clarity Captions



LDA-based
Topic Modeling



Main
Caption Topics

TOPIC MODELING ANALYSIS - RESULTS

LDA Topics Learned over high-level captions, $k = 15$

Assigned Label	Top 7 Words
“color options”	screen show app option color book differ
“login or create account”	user screen allow account log creat app
“select image from a list”	user screen allow select view list imag
“map search by location”	screen locat search map user show find

LDA Topics Learned over low-level captions, $k = 25$

Assigned Label	Top 7 Words
“page button”	page button top center bottom side left
“select date”	avail date select one option theme present
“camera button”	video imag photo pictur bottom camera
“privacy policy banner”	titl just term blue banner privaci polici

CLARITY PROJECT OVERVIEW

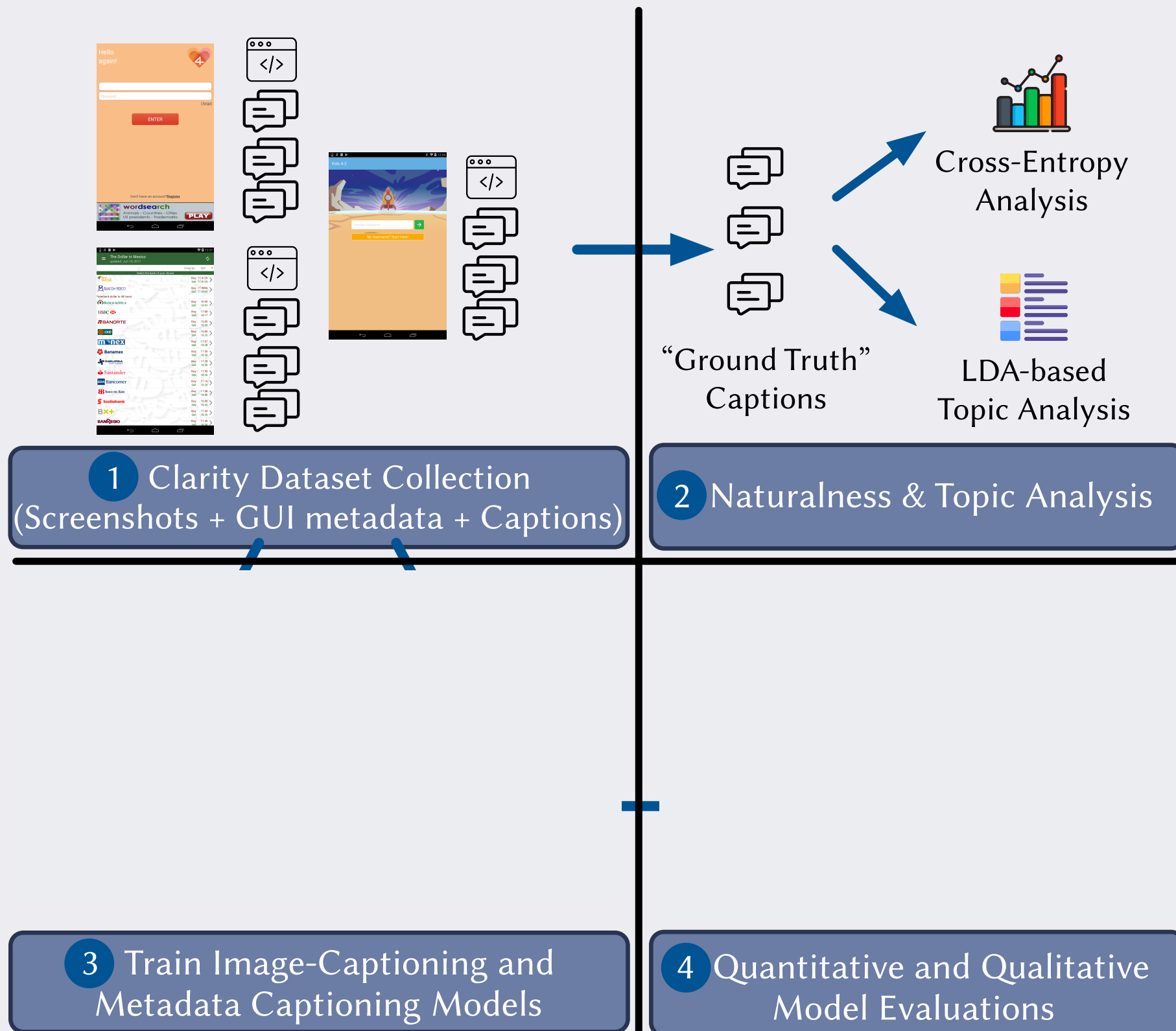


IMAGE AND METADATA CAPTIONING MODELS

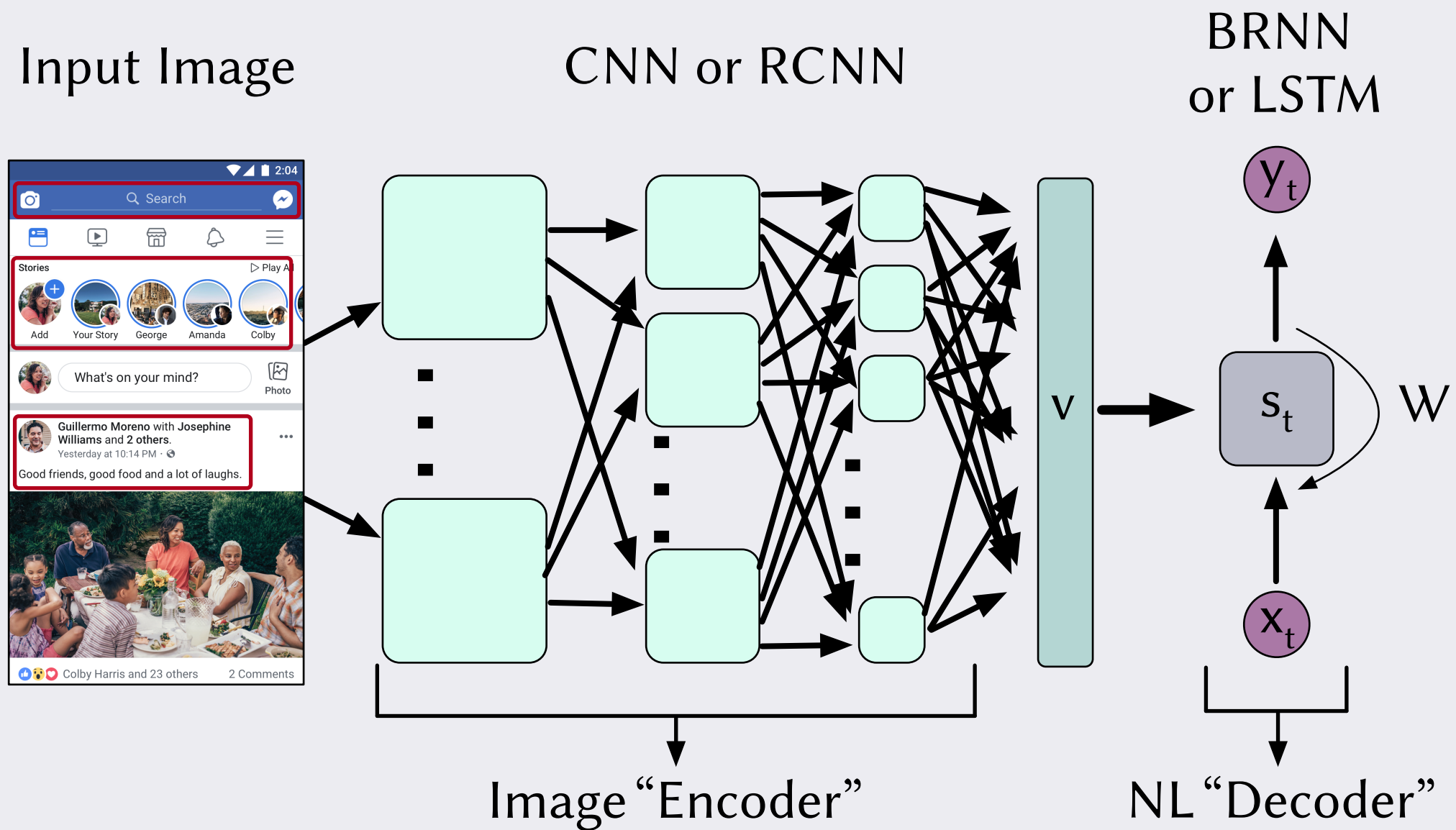
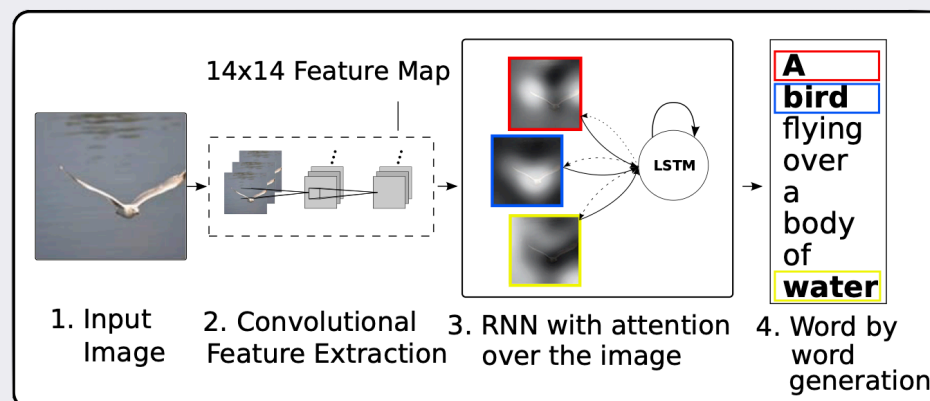
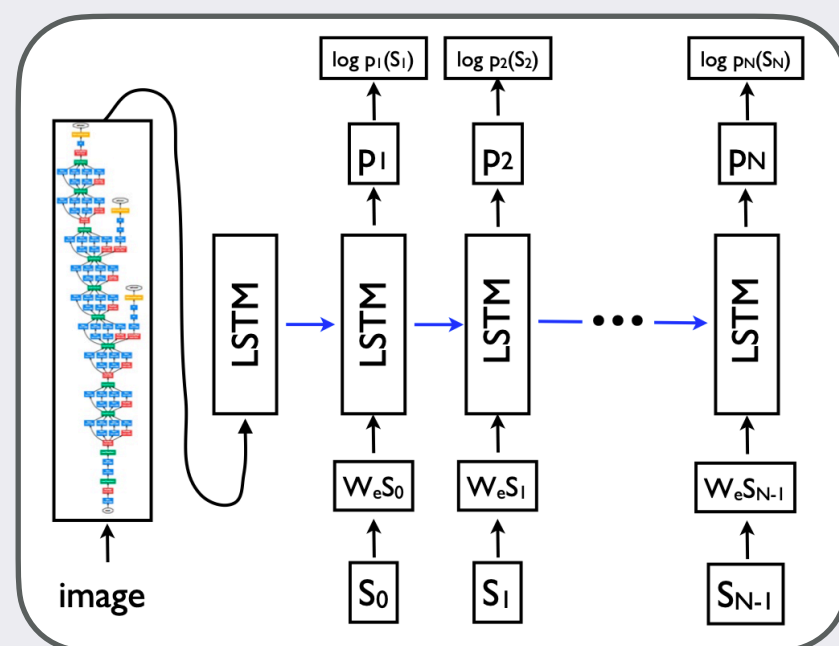


IMAGE CAPTIONING MODELS

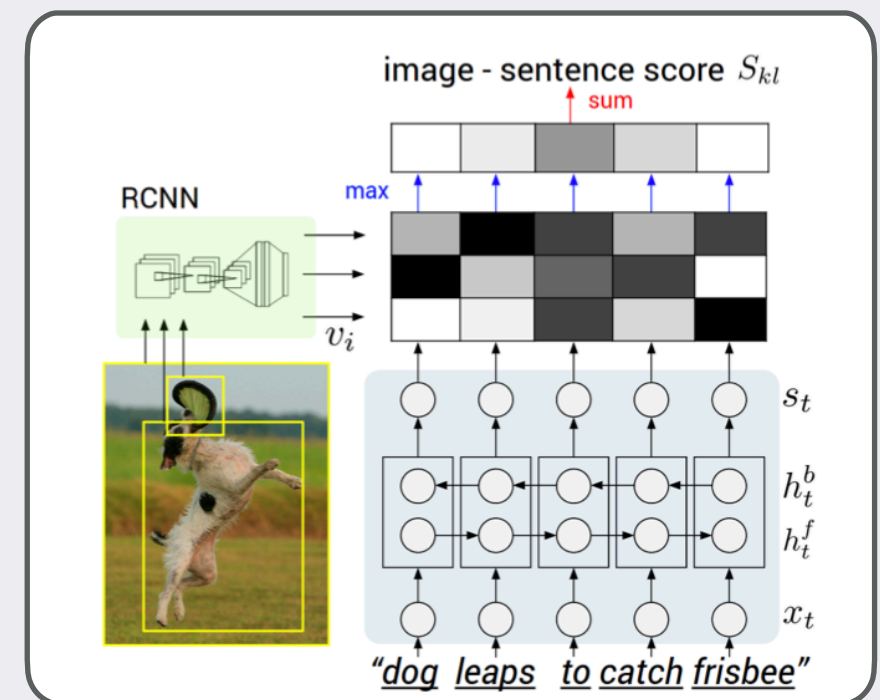
Three Base Models



Show, Attend & Tell



im2txt

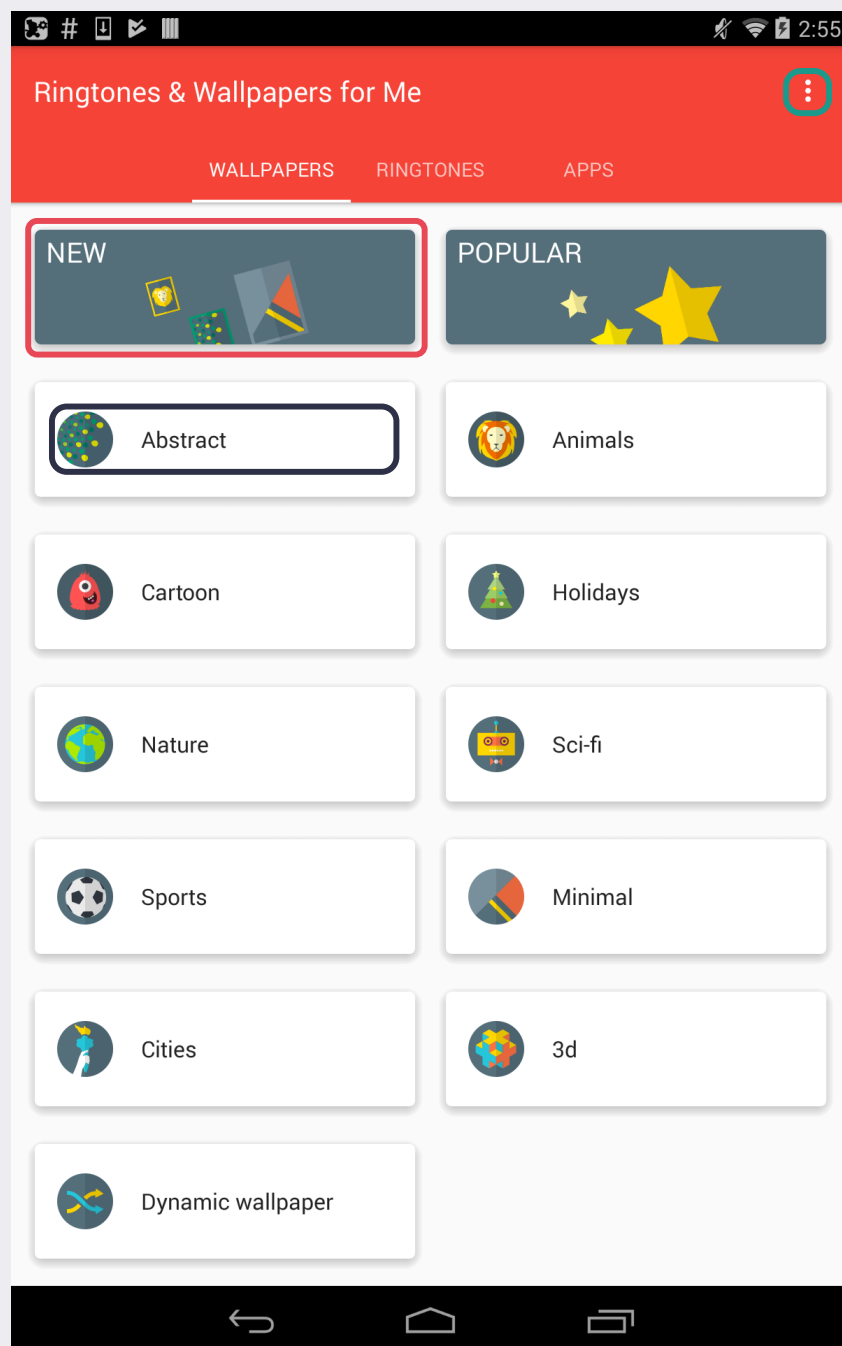


NeuralTalk2

IMAGE-CAPTIONING MODEL TREATMENTS

- Pre-trained on *general image dataset* (imagenet)
- Pre-trained on imagenet, *fine-tuned* on *ReDraw cropped* dataset
- Pre-trained on imagenet, *fine-tuned* on *ReDraw full-screen* dataset

METADATA CAPTIONING MODELS

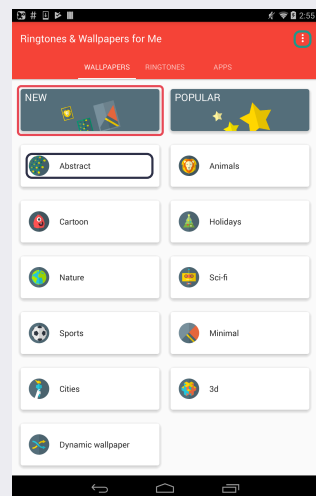


```
<node bounds="[1104,66][1184,162]"
checkable="false" checked="false"
class="android.widget.ImageView" clickable="true"
content-desc="More options" enabled="true"
focusable="true" focused="false" index="0" long-
clickable="true" package="com.apalon.ringtones"
password="false" scrollable="false"
selected="false" text=""/>
```

```
<node bounds="[34,313][578,477]" checkable="false"
checked="false" class="android.widget.ImageView"
clickable="false" content-desc="" enabled="true"
focusable="false" focused="false" index="0" long-
clickable="false" package="com.apalon.ringtones"
password="false" scrollable="false"
selected="false" text=""/>
```

```
<node bounds="[34,573][578,653]" checkable="false"
checked="false" class="android.widget.TextView"
clickable="false" content-desc="" enabled="true"
focusable="false" focused="false" index="0" long-
clickable="false" package="com.apalon.ringtones"
password="false" scrollable="false"
selected="false" text="Abstract"/>
```

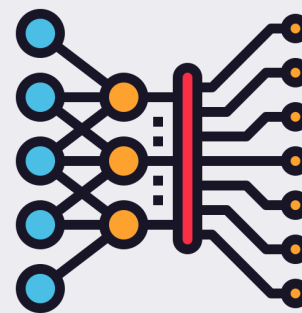
METADATA CAPTIONING MODELS



```
<node bounds="[1104,66][1184,162]" checkable="false" checked="false"
class="android.widget.ImageView" clickable="true"
content-desc="More options" enabled="true"
focusable="true" focused="false" index="0" long-
clickable="true" package="com.apalon.ringtones"
password="false" scrollable="false"
selected="false" text=""/>

<node bounds="[34,313][578,477]" checkable="false"
checked="false" class="android.widget.ImageView"
clickable="false" content-desc="" enabled="true"
focusable="false" focused="false" index="0" long-
clickable="false" package="com.apalon.ringtones"
password="false" scrollable="false"
selected="false" text=""/>

<node bounds="[34,573][578,653]" checkable="false"
checked="false" class="android.widget.TextView"
clickable="false" content-desc="" enabled="true"
focusable="false" focused="false" index="0" long-
clickable="false" package="com.apalon.ringtones"
password="false" scrollable="false"
selected="false" text="Abstract"/>
```



Clarity Metadata

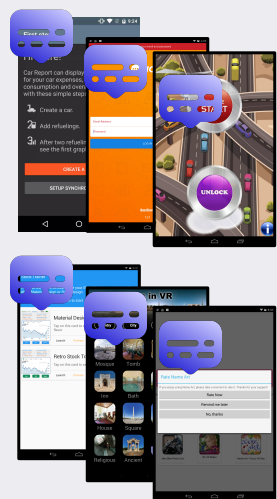
seq2seq
Encoder-Decoder
Model

Natural Language
Captions

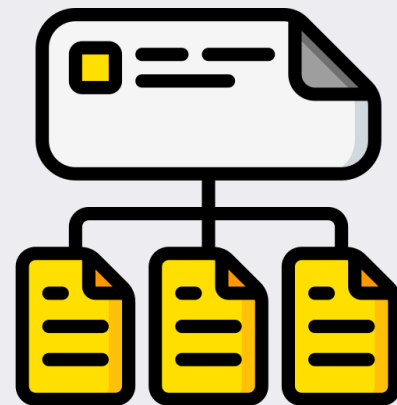
RESEARCH QUESTIONS

- **RQ₁:** Accuracy of Models?
- **RQ₂:** Accuracy, Completeness, & Understandability as rated by humans?

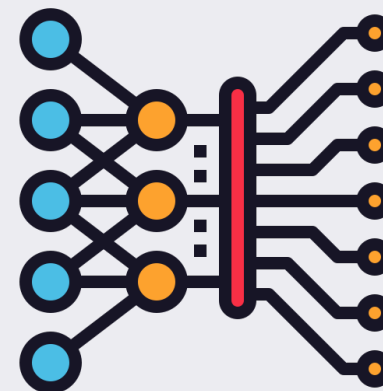
RQI: ACCURACY OF MODELS



Clarity
Dataset



Training/Validation/Test
Data Split



Neural Captioning
Models

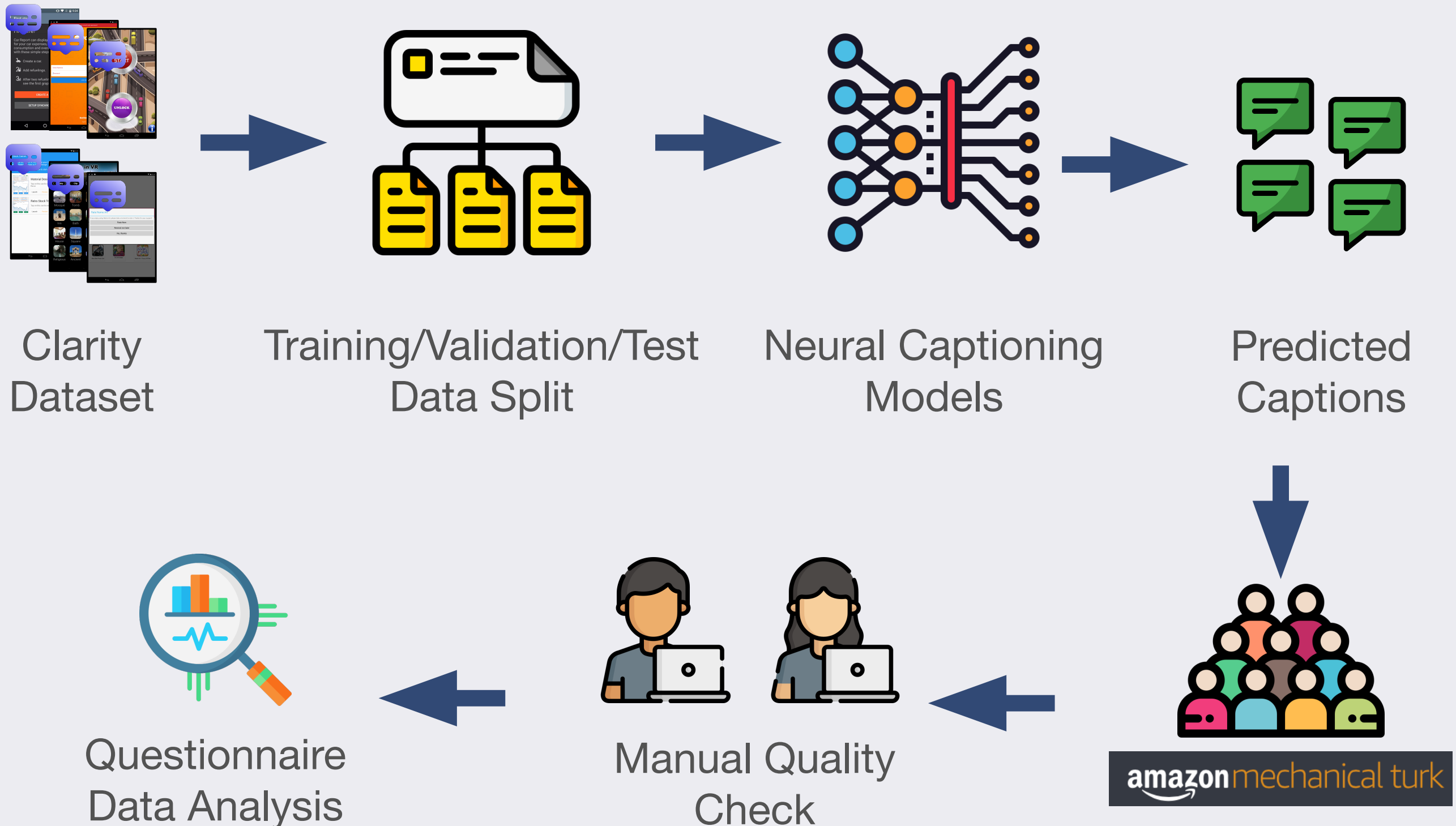


BLEU Scores

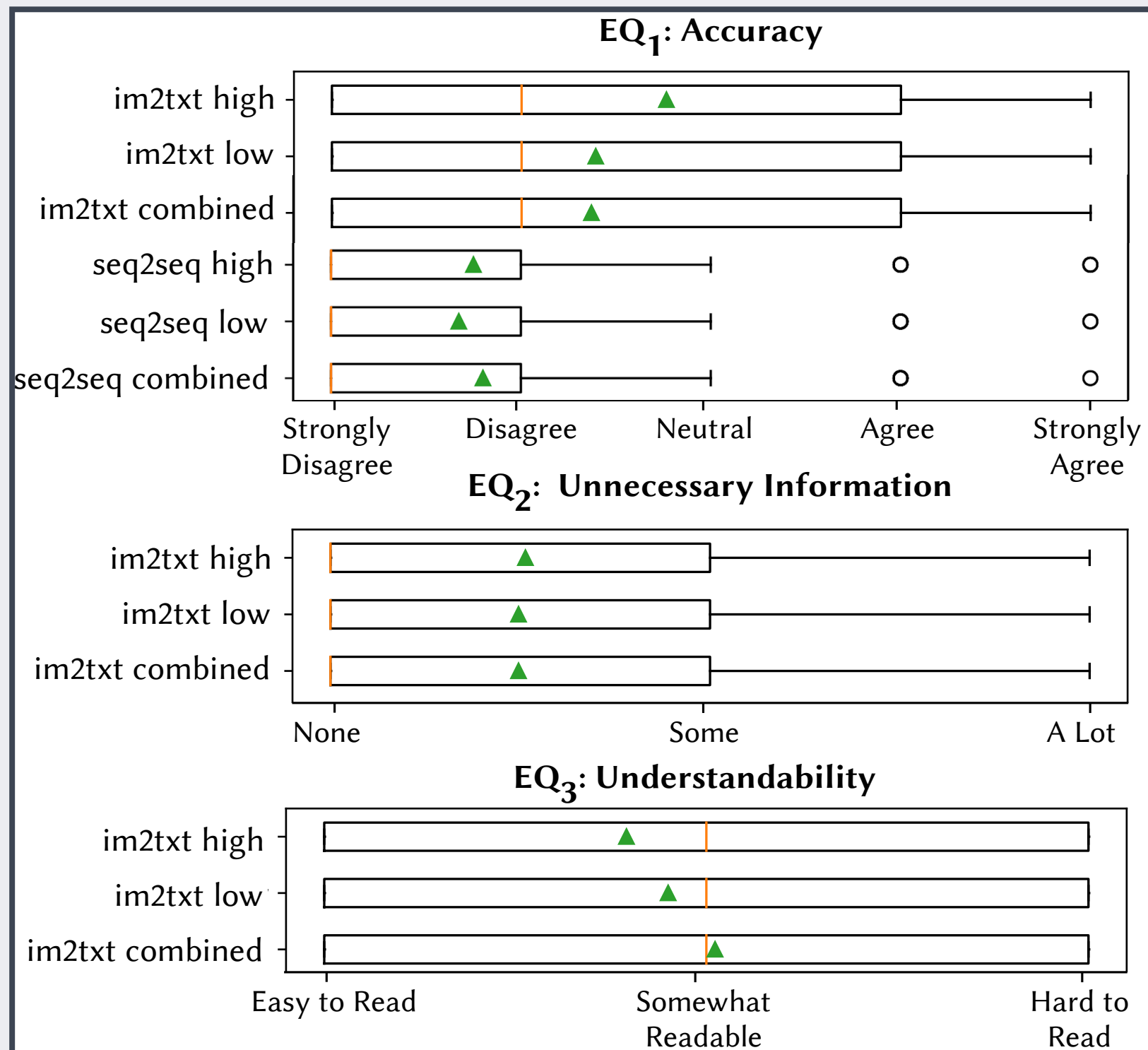
RQ1: RESULTS

Model	Capt.	Model Type	B _C	B ₁	B ₂	B ₃	B ₄
im2txt	High	im2txt-h-fs	12.4	24.8	12.6	6.7	5.3
	Low	im2txt-l-comp	27.0	45.6	31.8	20.0	10.1
	Comb.	im2txt-c-comp	30.3	51.7	35.9	22.1	11.6
NeuralTalk2	High	ntk2-h-imgnet	13.3	27.4	13.5	7.3	5.3
	Low	ntk2-l-ft	27.4	47.5	32.8	19.5	9.6
	Comb.	ntk2-c-ft	30.1	52.1	36.0	21.8	10.8
seq2seq	Low	seq2seq-l-type	18.1	44.6	17.0	7.9	0.24
	Comb.	seq2seq-c-type	16.9	38.9	14.7	6.0	0.08
SAT	High	sat-h	17.7	30.1	18.3	12.9	9.8
	Low	sat-l	35.0	52.5	38.7	28.1	20.7
	Comb.	sat-c	37.7	56.8	42.0	30.5	22.0
NeuralTalk2	Trained on Flickr8K		34.0	57.9	38.3	24.5	16.0
NeuralTalk2	Trained on MSCOCO		40.7	62.5	45.0	32.1	23.0
im2txt			42.6	66.6	46.1	32.9	24.6
SAT			45.7	71.8	50.4	35.7	25.0

RQ2: HUMAN RATING OF MODELS



RQ2: RESULTS



LESSONS LEARNED



Functional Descriptions
of GUIs exhibit a *high degree*
of *naturalness*



Our studied models
benefitted from *domain*
specific fine-tuning





Screenshots & Metadata
appear to have a degree
of *orthogonality*



It is difficult to predict
specific or *diverse*
pieces of functionality

OPEN SCIENCE

 An Empirical Investigation into the Use of Image Captioning for Automated Software Documentation 

Home

Authors

- **Kevin Moran**, George Mason University
- **Ali Yachnes**, William & Mary
- **George Purnell**, William & Mary
- **Junayed Mahmud**, George Mason University
- **Michele Tufano**, Microsoft
- **Carlos Bernal Cardenas**, Microsoft
- **Denys Poshyvanyk**, William & Mary
- **Zach H'Doubler**, William & Mary

Introduction

Graphical User Interface (GUI) based applications predominate user-facing software in the modern era of computing. High quality applications with well-designed GUIs allow users to instinctively understand underlying features. Thus, intuitively, certain functional properties of applications are encoded into the visual, pixel-based representation of the GUI such that cognitive human processes can determine the computing tasks provided by the interface. This suggests that there are certain latent *patterns* that exist within visual GUI data which indicate the presence of natural use cases capturing core program functionality. This functionality can be easily interpreted and communicated in natural language (NL) by users after simply glancing at a GUI.

Given the inherent representational power of GUIs in conveying program related information, we set forth the following hypothesis that serves as the basis for the work undertaken in this project:

The representational power of graphical user interfaces to convey program-related information can be meaningfully leveraged to support and automate software documentation tasks.

Table of contents

- Authors
- Introduction
- Background
 - The Connection between Images and NL
 - Mobile Graphical User Interfaces
- The CLARITY Dataset
 - Collecting the Clarity Dataset
 - Clarity Dataset Analysis
 - Results of Clarity Dataset Topic Modeling (RQ1)
 - Results of n-gram Language Modeling on the CLARITY Dataset (RQ2)
- Empirical Study
 - Image Captioning Model Configurations
 - Metadata Captioning Model Configurations
- Empirical Results
 - Results from Quantitative Model Evaluation (RQ3)
 - Predicted Captions for Best Model Configurations
 - Results from Qualitative Model Evaluation (RQ4)
- Dataset and Code

OPEN SCIENCE



<https://sagelab.io/Clarity/>