# Screenshot to Code

*Submitted in partial fulfilment of the requirements*

*of the degree of*

BACHELOR OF ENGINEERING
*in*
INFORMATION TECHNOLGY
(A.Y. 2018-2019)

by

Samdharsi Kumar (Roll No.: 17)
Esha Vijayvargiya (Roll No.: 56)
Harsh Vyas (Roll No.: 59)

Under the Guidance of
Mrs. Shruti Mathur
Assistant Professor, I.T Department, TCET

**University of Mumbai**

TCET
DEPARTMENT OF INFORMATION TECHNOLOGY (IT)
Credit Based Grading System [CBGS - 2012(R)]/Choice Based Credit and Grading Scheme [CBCGS - 2016(R)]
University of Mumbai

Estd. 2001

# DECLARATION

I/we declare that this written submission represents my/our ideas in my/our own words and where others ideas or words have been included, I/we have adequately cited and referenced the original sources. I/we also declare that I/we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my/our submission. I/we understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

1.Samdharsi Kumar (IT B-17)

2.Esha Vijayvargiya (IT B-56)

3.Harsh Vyas (IT B-59)

Date:

# ACKNOWLEDGEMENT

We sincerely thank our Principal, Dr. B.K. Mishra, Vice Principal, Dr. Deven Shah, Dean Dr. Kamal Shah, HOD, Dr. Rajesh Bansode for always encouraging us to do our best. We are highly indebted to our guide Mrs. Shruti Mathur who supported and constantly supervised us through this project and helped us in not only completing this project but also provided us with sample amount of knowledge that was really beneficial to us.

1. Samdharsi Kumar (Roll No.:17)

2. Esha Vijayvargiya (Roll No.:56)

3. Harsh Vyas (Roll No.:59)

# ABSTRACT

Neural machine translation is a recently proposed approach to machine translation. Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neural network that can be jointly tuned to maximize the translation performance. The models proposed recently for neural machine translation often belong to a family of encoder–decoders and encode a source sentence into a fixed-length vector from which a decoder generates a translation. In this paper, we conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture, and propose to extend this by allowing a model to automatically (soft-) search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. With this new approach, we achieve a translation performance comparable to the existing state-of-the-art phrase-based system on the task of English-to-French translation. Furthermore, qualitative analysis reveals that the (soft-) alignments found by the model agree well with our intuition.

Building on recent advances in image caption generation and optical character recognition (OCR), we present a general-purpose, deep learning-based system to decompile an image into presentational mark-up. While this task is a well-studied problem in OCR, our method takes an inherently different, data-driven approach. Our model does not require any knowledge of the underlying mark-up language, and is simply trained end-to-end on real-world example data. The model employs a convolutional network for text and layout recognition in tandem with an attention-based neural machine translation system. To train and evaluate the model, we introduce a new dataset of real-world rendered mathematical expressions paired with Latex mark-up, as well as a synthetic dataset of web pages paired with HTML snippets. Experimental results show that the system is surprisingly effective at generating accurate mark-up for both datasets.

# LIST OF FIGURES

# C O N T E N T S

# Chapter 1
# OVERVIEW

## 1.1 Introduction

Neural machine translation is a newly emerging approach to machine translation, recently proposed by Kalchbrenner and Blossom (2013), Sutskever et al. (2014) and Cho et al. (2014b). Unlike the traditional phrase-based translation system (see, e.g., Koehn et al., 2003) which consists of many small sub-components that are tuned separately, neural machine translation attempts to build and train a single, large neural network that reads a sentence and outputs a correct translation. Most of the proposed neural machine translation models belong to a family of encoder– decoders (Sutskever et al., 2014; Cho et al., 2014a), with an encoder and a decoder for each language, or involve a language-specific encoder applied to each sentence whose outputs are then compared (Hermann and Blossom, 2014). An encoder neural network reads and encodes a source sentence into a fixed-length vector. A decoder then outputs a translation from the encoded vector. The whole encoder–decoder system, which consists of the encoder and the decoder for a language pair, is jointly trained to maximize the probability of a correct translation given a source sentence.

A potential issue with this encoder–decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences, especially those that are longer than the sentences in the training corpus. Cho et al. (2014b) showed that indeed the performance of a basic encoder–decoder deteriorates rapidly as the length of an input sentence increases. In order to address this issue, we introduce an extension to the encoder–decoder model, which learns to align and translate jointly. Each time the proposed model generates a word in a translation, it (soft-) searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and not all the previous generated target words. To overcome such situation, we have come with an application, which would replace the doctors with self-diagnosis services. We are approaching towards this new future. As technology use increases, self-diagnosis using a symptom checker on the World Wide Web has become a topic of discussion in the health field.

## 1.2 Background

From a probabilistic perspective, translation is equivalent to finding a target sentence y that maximizes the conditional probability of y given a source sentence x, i.e., arg maxy p(y | x). In neural machine translation, we fit a parameterized model to maximize the conditional probability of sentence pairs using a parallel training corpus. Once a translation model learns the conditional distribution, given a source sentence a corresponding translation can be generated by searching for the sentence that maximizes the conditional probability.

Despite being a quite new approach, neural machine translation has already shown promising results. Sutskever et al. (2014) reported that the neural machine translation based on RNNs with long short-term memory (LSTM) units achieves close to the state-of-the-art performance of the conventional phrase-based machine translation system on an English-to-French translation task.1 Adding neural components to existing translation systems, for instance, to score the phrase pairs in the phrase table (Cho et al., 2014a) or to re-rank candidate translations has allowed to surpass the previous state-of-the-art performance level.

## 1.3 Importance of the project

The process of implementing client-side software based on a Graphical User Interface (GUI) mock-up created by a designer is the responsibility of developers. Implementing GUI code is, however, time-consuming and prevent developers from dedicating the majority of their time implementing the actual functionality and logic of the software they are building. Moreover, the computer languages used to implement such GUIs are specific to each target runtime system; thus resulting in tedious and repetitive work when the software being built is expected to run on multiple platforms using native technologies. In this paper, we describe a model trained end-to-end with stochastic gradient descent to simultaneously learns to model sequences and spatio-temporal visual features to generate variable-length strings of tokens from a single GUI image as input.

## 1.4 Perspective of Stakeholders and Customers

The automatic generation of programs using machine learning techniques is a relatively new field of research and program synthesis in a human-readable format have only been addressed very recently.

A recent example is Deep Coder, a system able to generate computer programs by leveraging statistical predictions to augment traditional search techniques. In another work by Gaunt et al., the generation of source code is enabled by learning the relationships between input-output examples via differentiable interpreters. Furthermore, Ling et al. recently demonstrated program synthesis from a mixed natural language and structured program specification as input. It is important to note that most of these methods rely on Domain Specific Languages (DSLs); computer languages (e.g. mark-up languages, programming languages, modelling languages) that are designed for a specialized domain but are typically more restrictive than full-featured computer languages. Using DSLs thus limit the complexity of the programming language that needs to be modelled and reduce the size of the search space. Although the generation of computer programs is an active research field as suggested by these breakthroughs, program generation from visual inputs is still a nearly unexplored research area. The closest related work is a method developed by Nguyen et al. to reverse-engineer Android user interfaces from screenshots. However, their method relies entirely on engineered heuristics requiring expert knowledge of the domain to be implemented successfully.

In order to exploit the graphical nature of our input, we can borrow methods from the computer vision literature. In fact, an important number of researches have addressed the problem of image captioning with impressive results; showing that deep neural networks are able to learn latent variables describing objects in an image and their relationships with corresponding variable-length textual descriptions. All these methods rely on two main components. First, a Convolutional Neural Network (CNN) performing unsupervised feature learning mapping the raw input image to a learned representation. Second, a Recurrent Neural Network (RNN) performing language modelling on the textual description associated with the input picture.

## 1.5 Objectives and Scope of the Project

The task of generating computer code written in a given programming language from a GUI screenshot can be compared to the task of generating English textual descriptions from a scene photography. In both scenarios, we want to produce variable-length strings of tokens from pixel values. We can thus divide our problem into three sub-problems. First, a computer vision problem of understanding the given scene (i.e. in this case, the GUI image) and inferring the objects present, their identities, positions, and poses (i.e. buttons, labels, element containers). Second, a language-modelling problem of understanding text (i.e. in this case, computer code) and generating syntactically and semantically correct samples. Finally, the last challenge is to use the solutions to both previous sub-problems by exploiting the latent variables inferred from scene understanding to generate corresponding textual descriptions (i.e. computer code rather than English) of the objects represented by these variables. We plan to offer various services parallelly.

## 1.6 Summary

Our first contribution is pix2code, a novel approach based on Convolutional and Recurrent Neural Networks allowing the generation of computer tokens from a single GUI screenshot as input. That is, no engineered feature extraction pipeline nor expert heuristics was designed to process the input data; our model learns from the pixel values of the input image alone. Our experiments demonstrate the effectiveness of our method for generating computer code for various platforms (i.e. iOS and Android native mobile interfaces, and multi-platform web-based HTML/CSS interfaces) without the need for any change or specific tuning to the model. In fact, pix2code can be used as such to support different target languages simply by being trained on a different dataset. A video demonstrating our system is available online1.

Our second contribution is the release of our synthesized datasets consisting of both GUI screenshots and associated source code for three different platforms. Our datasets and our pix2code implementation are publicly available2 to foster future research.

# Chapter 2
# LITERATURE SURVEY AND PROPOSED WORK (PHASE WISE)

## 2.1 Introduction

Our first contribution is Screenshot to Code, a novel approach based on Convolutional and Recurrent Neural Networks allowing the generation of computer tokens from a single GUI screenshot as input. That is, no engineered feature extraction pipeline nor expert heuristics was designed to process the input data; our model learns from the pixel values of the input image alone. Our experiments demonstrate the effectiveness of our method for generating computer code for various platforms (i.e. iOS and Android native mobile interfaces, and multi-platform web-based HTML/CSS interfaces) without the need for any change or specific tuning to the model. In fact, pix2code can be used as such to support different target languages simply by being trained on a different dataset. A video demonstrating our system is available online1. Our second contribution is the release of our synthesized datasets consisting of both GUI screenshots and associated source code for three different platforms. Our datasets and our pix2code implementation are publicly available2 to foster future research.

## 2.2 Problem definition (Phase Wise)

We have divided the project into 2 phases, which are further divided into 5 and 2 sub-phases each. The problem definitions for each phase and sub-phase are as follows.

**Phase I** – Phase I will focus on forming a base for the project. With thorough planning, analysis and design, we will ensure that the actual implementation is smoother.

1. Planning: Applying agile methodology for planning our Screenshot to Code project, and to achieve better results in the time frame given to us, along with better flexibility.
2. Analysis: Prepare an analysis of our own project. Prepare a detailed analysis on present software applications and overcoming its limitation and performance benchmarks.
3. Design: Integration of data and designing of event system application.
4. Coding: Writing the whole system app code and taking help of open source.
5. Implementation: Giving user to test beta testing and gathering centric analysis of performance, feedback, and try to improve the quality of the result/output.

**Phase II** – Phase II will focus on fine tuning the project and ensuring that the intended features will work as we wanted them to.

1. Testing: Doing the various test on system app like as Unit testing, Integration testing, Regression testing, system testing, etc., of test case data to check if the Integrated system functions are as desired by the client.
2. Deployment: Conduct beta testing for identifying any further errors, bugs and improvements that can be performed. After testing and approval, deploy the proposed system.

## 2.3 Literature Survey Table

| Reference No. | Paper Title | Author | Year of Publication | Key Findings | Research Gaps |
|---|---|---|---|---|---|
| 1 | Pix2code: Generating Code from a Graphical User Interface Screenshot | Tony Beltramelli | 2016 | Transforming a graphical user interface screenshot created by a designer into computer code is a typical task conducted by a developer in order to build customized software. | The code, which is generated from the screenshot, is restricted to only few coding platforms. |
| 2 | What You Get Is What You See: A Visual Markup Decompiler | Yuntian Deng, Anssi Kanervisto, Alexander M. Rush | 2015 | Building on recent advances in image caption generation and optical character recognition (OCR). | Standard mode of language is required for the generation of code. |
| 3 | Neural machine translation by jointly learning to align and translate | Dzmitry Bahdanau | 2017 | Neural machine translation is a recently proposed approach to machine translation. Unlike the traditional statistical machine translation. | The time required to generate code from the screenshot depends upon the complexity of the webpage. |

## 2.4 Methodology Used

We define the image-to-mark-up problem as converting a rendered source image to target presentational mark-up that fully describes both its content and layout. The source, $x \in X$ , consists of an image with height H and width W, e.g. R H×W for grayscale inputs. The target, $y \in Y$, consists of a sequence of tokens y1, y2, · · ·, yC where C is the length of the output, and each y is a token in the mark-up language with vocabulary Σ. The rendering is defined by a possibly unknown, many-to-one, compile function, compile: $Y \rightarrow X$ . In practice this function may be quite complicated, e.g. a browser, or ill-specified, e.g. the LaTeX language. The supervised task is to learn to approximately invert the compile function using supervised examples of its behaviour. We assume that we are given instances (x (1) , y (1)), ...,(x (J) , y (J) ), with possibly differing dimensions H, W, C and that, compile(y) ≈ x, for all training pairs (x, y). At test time, the system is given a raw input x rendered from ground-truth y. It generates a hypothesis yˆ that can then be rendered by the black-box function xˆ = compile(yˆ). Evaluation is done between xˆ and x, i.e. the aim is to produce similar rendered images while yˆ may or may not be similar to the ground-truth mark-up y.

Each row is then encoded using a recurrent neural network (RNN). These encoded features are then used by an RNN decoder with a visual attention mechanism. The decoder implements a conditional language model over the vocabulary Σ, and the whole model is trained to maximize the likelihood of the observed mark-up. The full structure is illustrated in Figure 2. We describe the model in more detail. Convolutional Network The visual features of an image are extracted with a multi-layer convolutional neural network interleaved with max-pooling layers. This network architecture is now standard; we model it specifically after the network used by Shi et al. (2015) for OCR from images. The CNN takes the raw input R H×W and produces a feature grid V of size D × H0 × W0 , where c denotes the number of channels and H0 and W0 are the reduced sizes from pooling. Row Encoder In attention-based image captioning, the image feature grid can be directly fed into the decoder.

## 2.5 Summary

The turn towards data-driven neural methods for image and text leads us to revisit the problem of generating structured mark-up. We consider whether a supervised model can learn to produce correct presentational mark-up from an image, without requiring a textual or visual grammar of the underlying mark-up language. While results from language modelling suggest that neural models can consistently generate syntactically correct mark-up (Karpathy, Johnson, and Li 2015; Vinyals et al. 2015a), it is unclear whether the full solution can be learned from mark-up-image pairs. Our model, WYGIWYS [What you get is what you see], is a simple extension of the attention-based encoder-decoder model (Bahdanau, Cho, and Bengio 2014), which is now standard for machine translation. Similar to work in image captioning (Xu et al. 2015), the model incorporates a multi-layer convolutional network over the image with an attention-based recurrent neural network decoder.

# Chapter 3

# ANALYSIS AND PLANNING

## 3.1 Introduction

The analysis phase is the most crucial phase of any project. The quality of the analysis can make or break a project. Machine Learning presents challenges due the complexity involved in getting the balance between too much and not enough. Planning such a task is matter of finding what we really need and we do not.

We may design systems which can process an extremely wide variety of inputs, but we cannot actively ensure that the AI will respond to it in the way we want it to. It may present us an output which was intended to be for another input, or it may not be able to process it due to the load caused by the heavy processing on the interpreter. We have to plan for such a scenario too. Thus, the challenge is not just of input handling or building features, but of efficiency as well. We did this phase slowly so we be could be thorough with all our requirements and plans. A slow approach also allowed us to anticipate risks. Planning phase is probably the best time to plan for risks and avoid them altogether. Being careful in the planning phase allows us to put less effort into the risk mitigation, risk monitoring and risk management plan.

## 3.2 Feasibility Study

Economic feasibility

The project relies on open source software, such as Python and HTML. The functionality provided by these software's is enough to construct a product capable of usage for required applications.

For the machine's learning and training, we will use freely available datasets. These sets run into thousands of lines, and thus can provide enough learning capability to the software. Due to these two being freely available, the project is very low cost and thus feasible from an economic standpoint.

The expected total cost is thus 0 rupees. As such, we do not need any outside funding. The project will be expected to make minimal sales, due to the presence of a variety of other similar tool and the slow adaption of software's by the technical community.

Technical feasibility

The capabilities required for the project are fairly in the feasible range. Most of the planned feature hinge around processing datasets. A GUI is required, which will be done with the pyGUI framework. The GUI will be kept simple and minimalist. We also have a feature for users to set personal data, preferences.

For the machine learning, we will use Machine Learning Algorithm to train our datasets, both of which are available on open source repositories.

Operational feasibility

On an operational level, the software will need to run intensive processing tasks while it's in its learning phase. Since the learning phase is a very short process, it is doable without the need for Extra hardware. The software will be deployed as a Desktop application and will need users to download it. It will also need internet to operate.

## 3.3 Project planning (Resources, Tools used, etc.)

IT projects require resources in terms of money, time, human resources, infrastructure and technology, both hardware and software. Resources are not just a mean, but also an approximation of constraints. Stakeholder perspective is crucial to the success of this project. Part of the reason is that medical diagnosis is a highly sensitive field, and even the slightest of errors, which are evidently unavoidable in even the most sophisticated software, can lead to the patient's condition worsening. We divided our project into various phases and sub-phases, and allocated date ranges from a week to 3 weeks to every sub-phase. This was done using the timeline chart feature of online MS project. We then used a Gantt chart to model schedule dependencies and fine tune the scheduling. The Gantt chart was created using the Gantt chart feature available in MS Excel. Using these two, we further planned our sub-phases.

## 3.4 Scheduling (Time line chart or Gantt chart) according to sprint backlog

| **Task** | **Duration** | **No. of days** |
|---|---|---|
| Project Title Finalization | 14/01/2019 - 16/01/2019 | 1 |
| Literature survey | 19/01/2019 - 30/01/2019 | 4 |
| Business case | 02/02/2019 - 05/02/2019 | 1 |
| Project Charter | 08/02/2019 - 10/02/2019 | 2 |
| Requirement gathering | 13/02/2019 - 17/02/2019 | 1 |
| Security planning | 20/02/2019 - 24/02/2019 | 2 |
| Legal planning and user survey | 26/02/2019 - 28/02/2019 | 3 |
| Implementing and testing of basic functionality | 3/03/2019 - 14/03/2019 | 2 |
| Implementation and testing of GUI | 17/03/2019 - 21/03/2019 | 3 |
| Implementing and testing of machine learning functionality | 24/03/2019 - 5/04/2019 | 3 |
| Synopsis and report | 8/04/2019 – 18/04/2019 | 1 |
| Final presentation and viva | 22/04/2019 - 23/04/2019 | 2 |

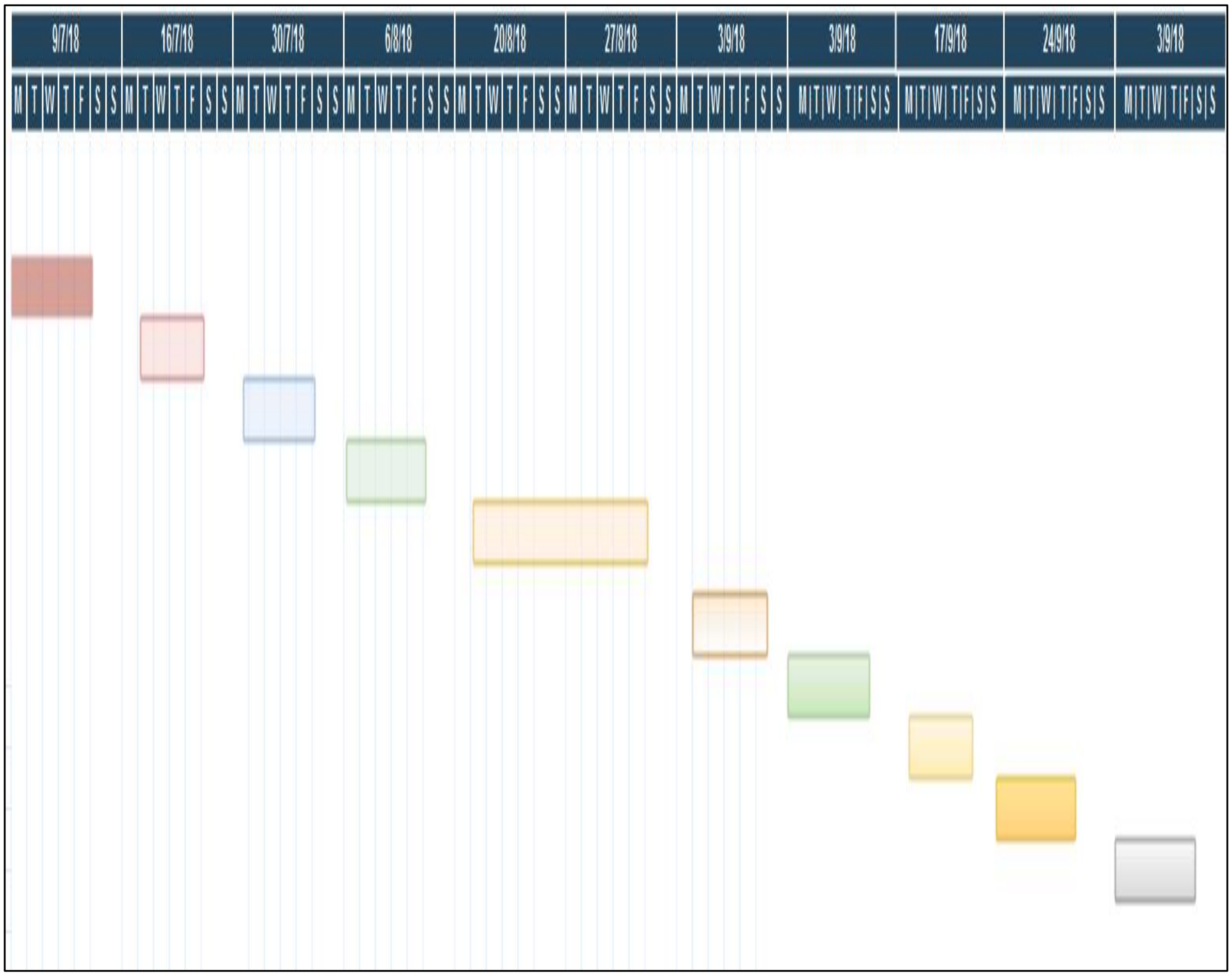


Fig 3.4 Gantt Chart

## 3.5 Summary

Here we summarize the analysis and planning phase of our project. The phase included scheduling, budgeting and creation of subtasks. The subtasks help us properly define our needs and features.

The analysis phase is the most crucial phase of any project. The quality of the analysis can make or break a project. Artificial intelligence presents challenges due the complexity involved in getting the balance between too much and not enough. Planning such a task is matter of finding what we really need and we don't.

We did this phase slowly so we be could be thorough with all our requirements and plans. A slow approach also allowed us to anticipate risks. The tasks, which were already done, were - analysis, design, modelling, literature survey, budgeting and scheduling, feasibility analysis.

The scheduling included a plan for the first half of the year. The plan includes - project title finalization, literature survey, business case, project charter, requirement gathering, security planning, legal planning and user survey, implementation and testing of basic functionality, implementation and testing of GUI, implementation and testing of machine learning functionality, synopsis and report and final presentation.

# Chapter 4

# DESIGN AND IMPLEMENTATION

## 4.1 DFD

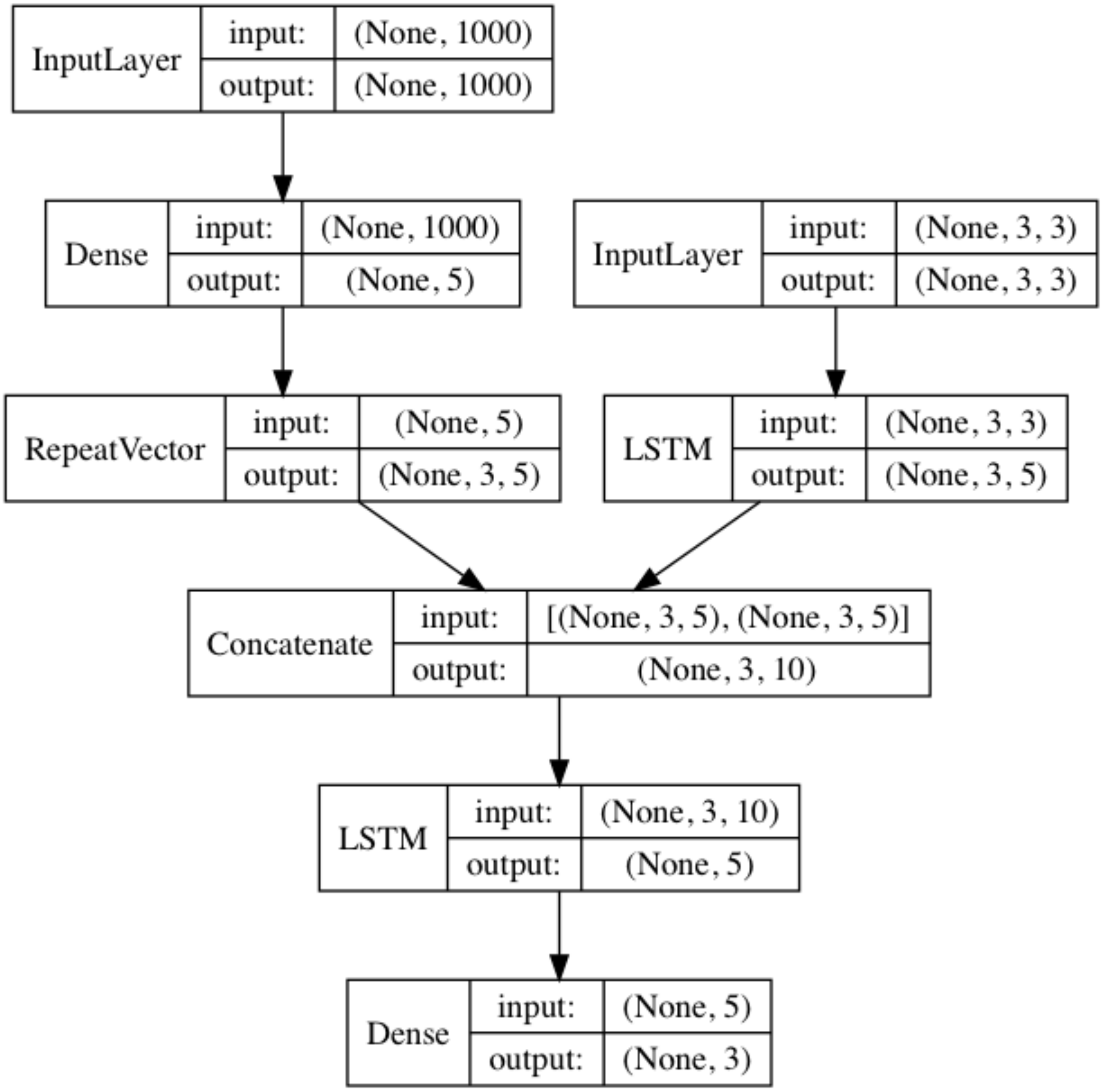


Fig 4.1 Data Flow Diagram

### 4.2. Block Chart Diagram

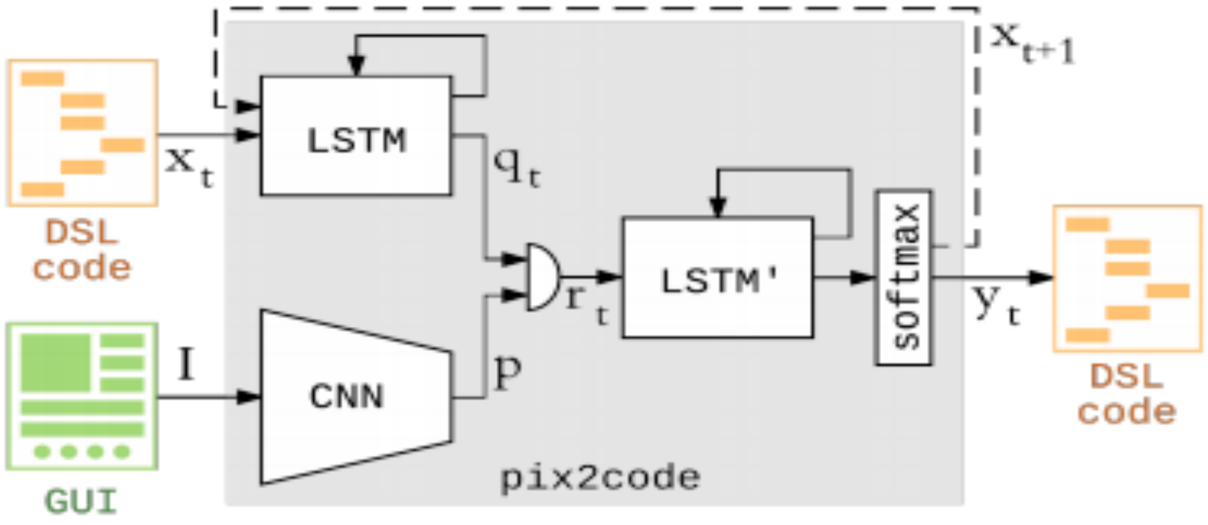
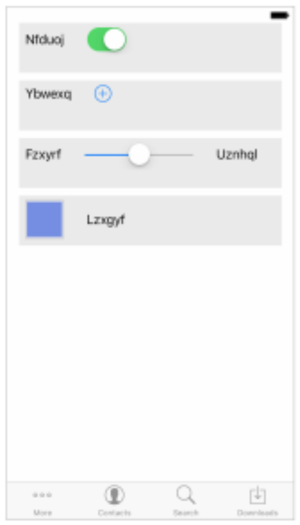


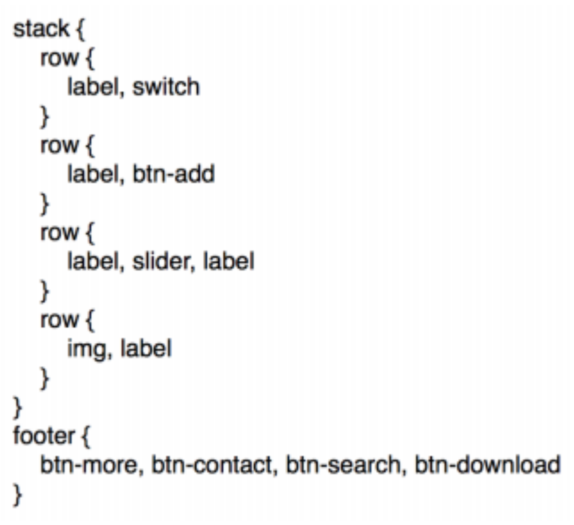
Fig 4.2 Overview of the pix2code model architecture

During training, the GUI image is encoded by a CNN-based vision model; the context (i.e. a sequence of one-hot encoded tokens corresponding to DSL code) is encoded by a language model consisting of a stack of LSTM layers. The two resulting feature vectors are then concatenated and fed into a second stack of LSTM layers acting as a decoder. Finally, a softmax layer is used to sample one token at a time; the output size of the softmax layer corresponding to the DSL vocabulary size. Given an image and a sequence of tokens, the model (i.e. contained in the gray box) is differentiable and can thus be optimized end-to-end through gradient descent to predict the next token in the sequence. During sampling, the input context is updated for each prediction to contain the last predicted token. The resulting sequence of DSL tokens is compiled to the desired target language using traditional compiler design techniques.

## 4.3. GUI



(a) iOS GUI screenshot



```
stack {
   row {
      label, switch
   }
   row {
      label, btn-add
   }
   row {
      label, slider, label
   }
   row {
      img, label
   }
}
footer {
   btn-more, btn-contact, btn-search, btn-download
}
```

(b) Code describing the GUI written in our DSL

Fig 4.3 GUI Implementation

CNNs are currently the method of choice to solve a wide range of vision problems thanks to their topology allowing them to learn rich latent representations from the images they are trained on [16, 11]. We used a CNN to perform unsupervised feature learning by mapping an input image to a learned fixed-length vector; thus acting as an encoder as shown in Figure 1. The input images are initially re-sized to 256 × 256 pixels (the aspect ratio is not preserved) and the pixel values are normalized before to be fed in the CNN. No further pre-processing is performed. To encode each input image to a fixed-size output vector, we exclusively used small 3 × 3 receptive fields which are convolved with stride 1 as used by Simonyan and Zisserman for VGGNet [18]. These operations are applied twice before to down-sample with max-pooling. The width of the first convolutional layer is 32, followed by a layer of width 64, and finally width 128. Two fully connected layers of size 1024 applying the rectified linear unit activation complete the vision model.

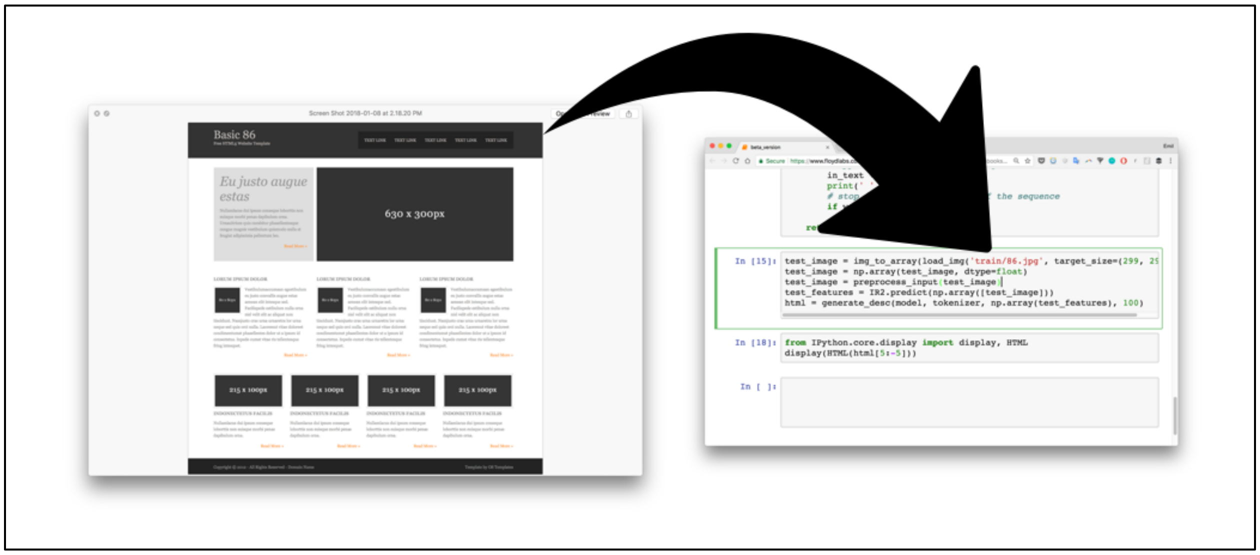
# Chapter 5

# RESULTS AND DISCUSSION

## 5.1 Actual Results

Outputs:

We presented Screenshot to Code, a novel method to generate computer code given a single GUI image as input. While our work demonstrates the potential of such a system to automate the process of implementing GUIs, we only scratched the surface of what is possible. Our model consists of relatively few parameters and was trained on a relatively small dataset. The quality of the generated code could be drastically improved by training a bigger model on significantly more data for an extended number of epochs. Implementing a now-standard attention mechanism [1, 22] could further improve the quality of the generated code. Using one-hot encoding does not provide any useful information about the relationships between the tokens since the method simply assigns an arbitrary vectorial representation to each token. Therefore, pre-training the language model to learn vectorial representations would allow the relationships between tokens in the DSL to be inferred (i.e. learning word embedding such as word2vec [13]) and as a result alleviate semantical error in the generated code. Furthermore, one-hot encoding does not scale to very big vocabulary and thus restrict the number of symbols that the DSL can support.



```
In [15]: test_image = img_to_array(load_img('train/86.jpg', target_size=(299, 29
         test_image = np.array(test_image, dtype=float)
         test_image = preprocess_input(test_image)
         test_features = IR2.predict(np.array([test_image]))
         html = generate_desc(model, tokenizer, np.array(test_features), 100)

In [18]: from IPython.core.display import display, HTML
         display(HTML(html[5:-5]))

In [ ]:
```

Fig 5.1(a) Project Implementation

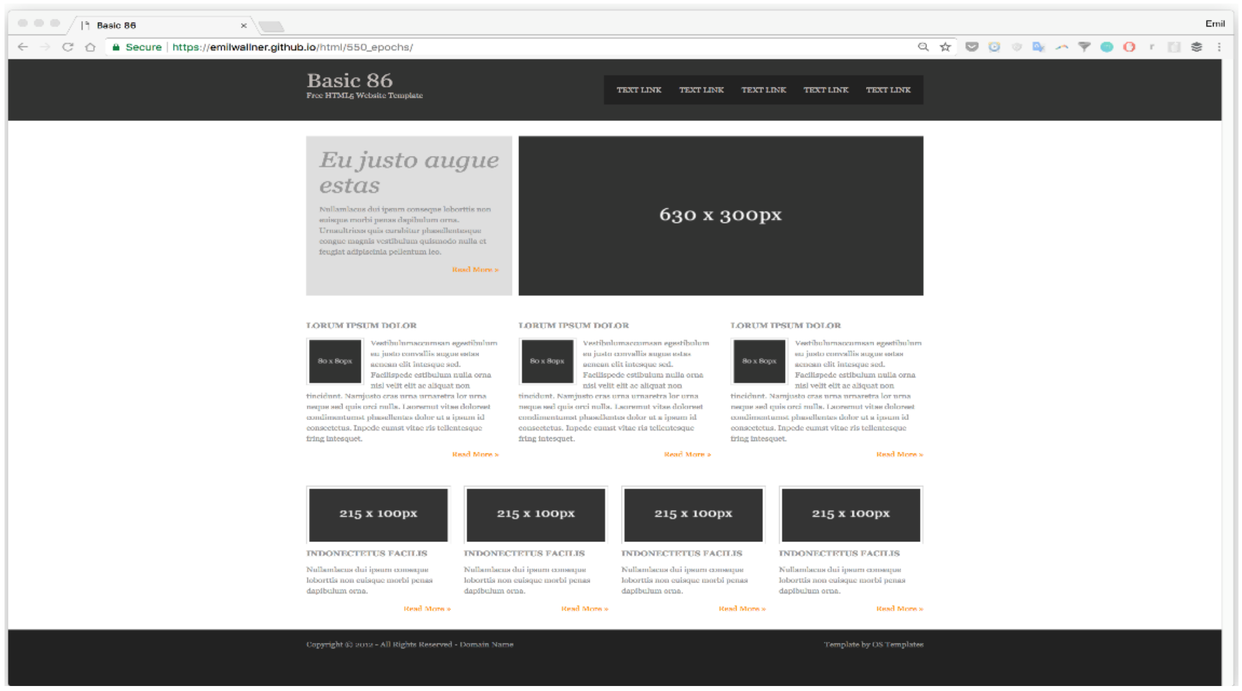


Fig 5.1(b) Executing the Generated Code from the Screenshot

Generative Adversarial Networks GANs have shown to be extremely powerful at generating images and sequences. Applying such techniques to the problem of generating computer code from an input image is so far an unexplored research area. A major drawback of deep neural networks is the need for a lot of training data for the resulting model to generalize well on new unseen examples. One of the significant advantages of the method we described in this paper is that there is no need for human-labelled data. In fact, the network can model the relationships between graphical components and associated tokens by simply being trained on image-sequence pairs. Although we used data synthesis in our paper partly to demonstrate the capability of our method to generate GUI code for various platforms; data synthesis might not be needed at all if one wants to focus only on web-based GUIs. In fact, one could imagine crawling the World Wide Web to collect a dataset of HTML/CSS code associated with screenshots of rendered websites. Considering the large number of web pages already available online and the fact that new websites are created every day, the web could theoretically supply a virtually unlimited amount of 7 training data; potentially allowing deep learning methods to fully automate the implementation of web-based GUIs.

## 5.2 Future Scope

Possible future directions for this work include: scaling the system to run on full websites or for document DE compilation, using similar approach for handwritten mathematical expressions or HTML from informal sketches, or combining these methods with neural inference machines such as MemNNs (Weston, Chopra, and Bordes 2014) for more complicated mark-up or reference variables.

## 5.3 Testing

One of the most overlooked (or ignored) aspects of building a Machine Learning model is to check whether the data used for training and testing the model are sanitized or if they belong to an adversary data set. The adversary data sets are the ones that can be used to skew the results of the model by training the model using incorrect data. This is also termed as data poisoning attacks. The other two techniques we will be using are:

Supervised Learning

Regression: Regression models are used to make numerical predictions. For example, what would be the price of the stock on a given day?

Classification: Classification models are used to predict the class of a given data. For example, whether a person is suffering from a disease or not.

Unsupervised Learning

Clustering: Clustering models are used to learn different classes (cluster) from a given data set without being fed with any kind of label information, unlike supervised learning. Once the model is learned, the model is used to predict the class of the new data set. For example, grouping news in different classes and associating the labels with the learned classes.

The data sets consist of more than ten thousand entries. First, the initial 80% of the data will be used for training the model and the rest 20% will be used for testing. Then, the first 20% and last 20% will be used for testing the model and the middle 60% for training the model.

The other techniques we will also incorporate are:

1.Model performance

2.Metamorphic testing

3.Dual coding

4.Coverage guided fuzzing

5.Comparison with simplified, linear models

6.Testing with different data slices

## 5.4 Deployment

Deployment can be a major challenge in software which involves machine learning. Huge amounts of processing is required just to get it up and running. Data generated in the process is humongous as well.

The ML model will be deployed using Google firebase or spring. The cloud platform will provide a base for future developments where flexibility, portability and reliability is required.

The GUI by itself is not very difficult to deploy. We have multiple ways of offering int erfaces, such as web interfaces and command line interfaces. It will expand to a desktop GUI in the future as well.

# Chapter 6
# CONCLUSION

## Conclusion:

The conventional approach to neural machine translation, called an encoder–decoder approach, encodes a completely input sentence into a fixed-length vector from which a translation will be decoded. We conjectured that the use of a fixed-length context vector is problematic for translating long sentences, based on a recent empirical study reported by Cho et al. and Pouget-Abadie et al.

We proposed a novel architecture that addresses this issue. We extended the basic encoder–decoder by letting a model (soft-) search for a set of input words, or their annotations computed by an encoder, when generating each target word. This frees the model from having to encode a whole source sentence into a fixed-length vector, and lets the model focus only on information relevant to the generation of the next target word. This has a major positive impact on the ability of the neural machine translation system to yield good results on longer sentences. Unlike with the traditional machine translation systems, all of the pieces of the translation system, including the alignment mechanism, are jointly trained towards a better log-probability of producing correct translations. We tested the proposed model, called RNNsearch, on the task of English-to-French translation. The experiment revealed that the proposed RNNsearch outperforms the conventional encoder–decoder model (RNNencdec) significantly, regardless of the sentence length and that it is much more robust to the length of a source sentence. From the qualitative analysis where we investigated the (soft-) alignment generated by the RNNsearch, we were able to conclude that the model can correctly align each target word with the relevant words, or their annotations, in the source sentence as it generated a correct translation. Perhaps more importantly, the proposed approach achieved a translation performance comparable to the existing phrase-based statistical machine translation.

## References:

**1.** D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.

**2.** M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. Deepcoder: Learning to write programs. arXiv preprint arXiv:1611.01989, 2016.

**3.** B. Dai, D. Lin, R. Urtasun, and S. Fidler. Towards diverse and natural image descriptions via a conditional gan. arXiv preprint arXiv:1703.06029, 2017.

**4.** J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2625–2634, 2015.

**5.** A. L. Gaunt, M. Brockschmidt, R. Singh, N. Kushman, P. Kohli, J. Taylor, and D. Tarlow. Terpret: A probabilistic programming language for program induction. arXiv preprint arXiv:1608.04428, 2016.