BigDat 2020
Ancona, Italy

BIG CODE

(Part 1/3)

Dr. Jayanti Prasad

6th Winter school on Big Data

(13-18 January, 2020)

Ancona, ITALY

OBJECTIVES & DISCLAIMERS

Objectives

- Introduce a new & emerging field of machine learning applications.
- To present a roadmap which can be easily followed (Big-Code).
- Review some recent results in the field of machine learning on source code.

Disclaimers

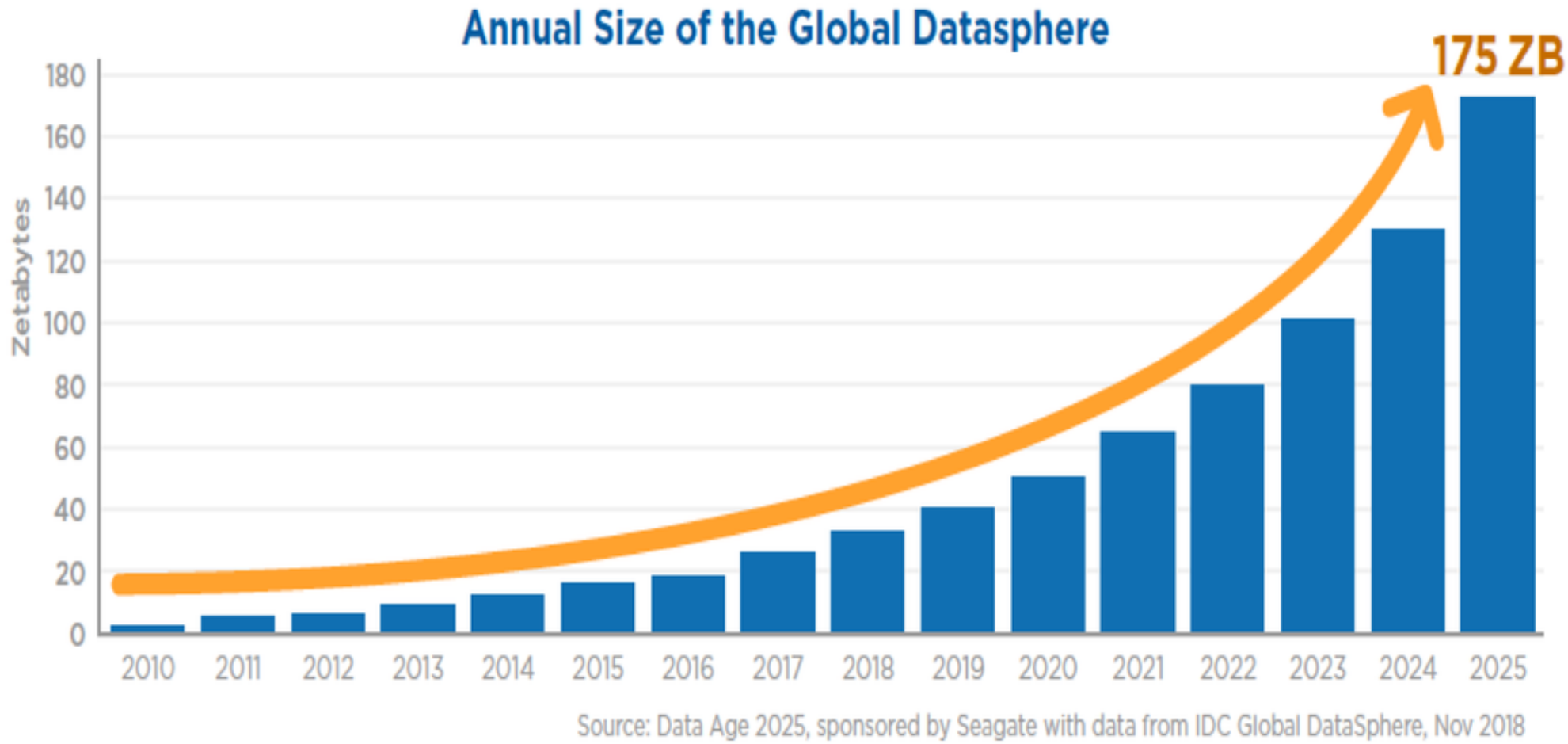
- What is presented is a roadmap & not any concrete implementation.
- All the resources which are used are open source & referenced.
- In order to evaluate the approach it needs to be tried on large data sets with large systems.

4TH INDUSTRIAL REVOLUTION

Driving Forces

- Data explosion
- Affordable computing & communication
- Artificial Intelligence & Machine Learning

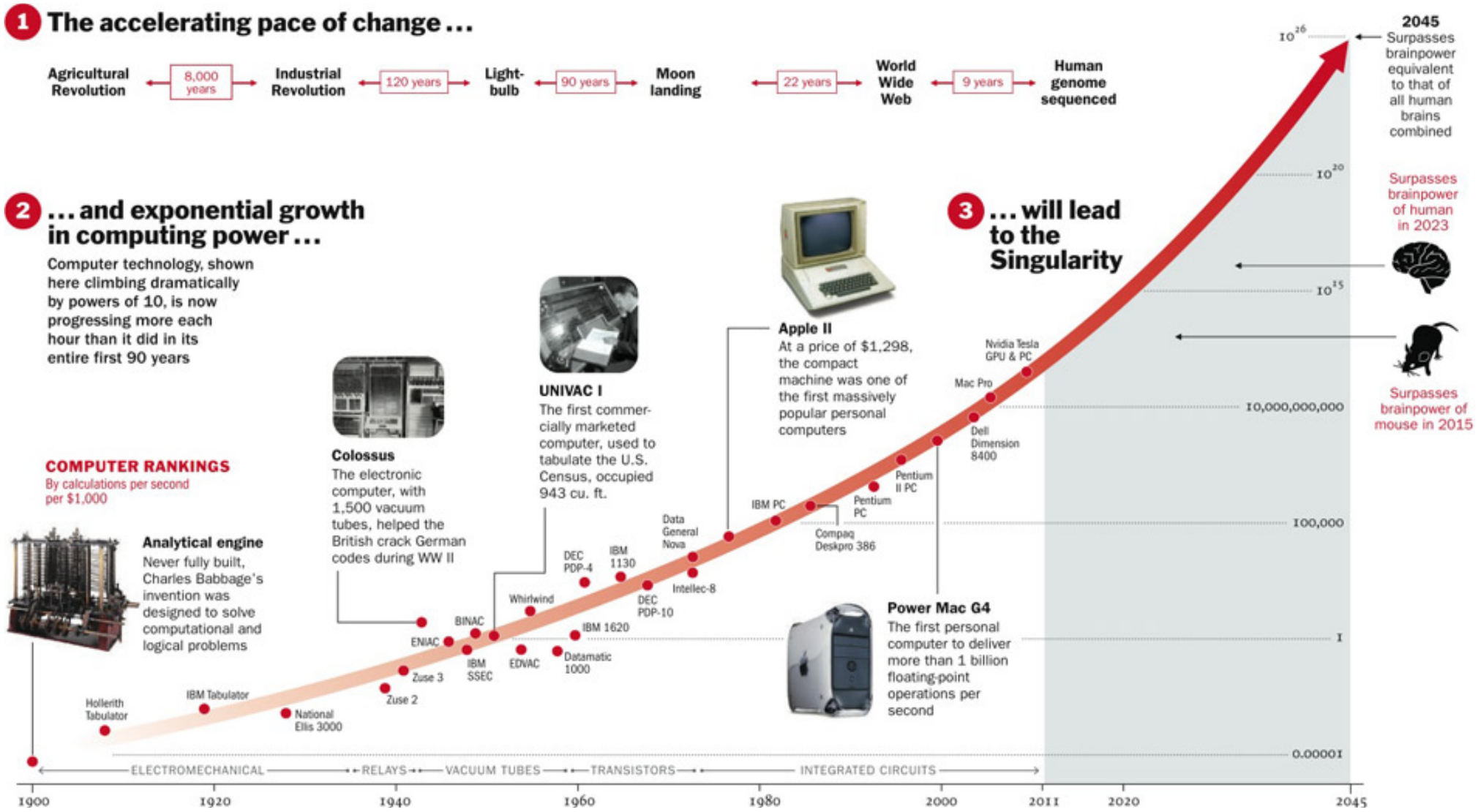
GROWTH OF DATA



Reference : Tom Coughlin (2018) 175 Zettabytes by 2025 [Forbes]

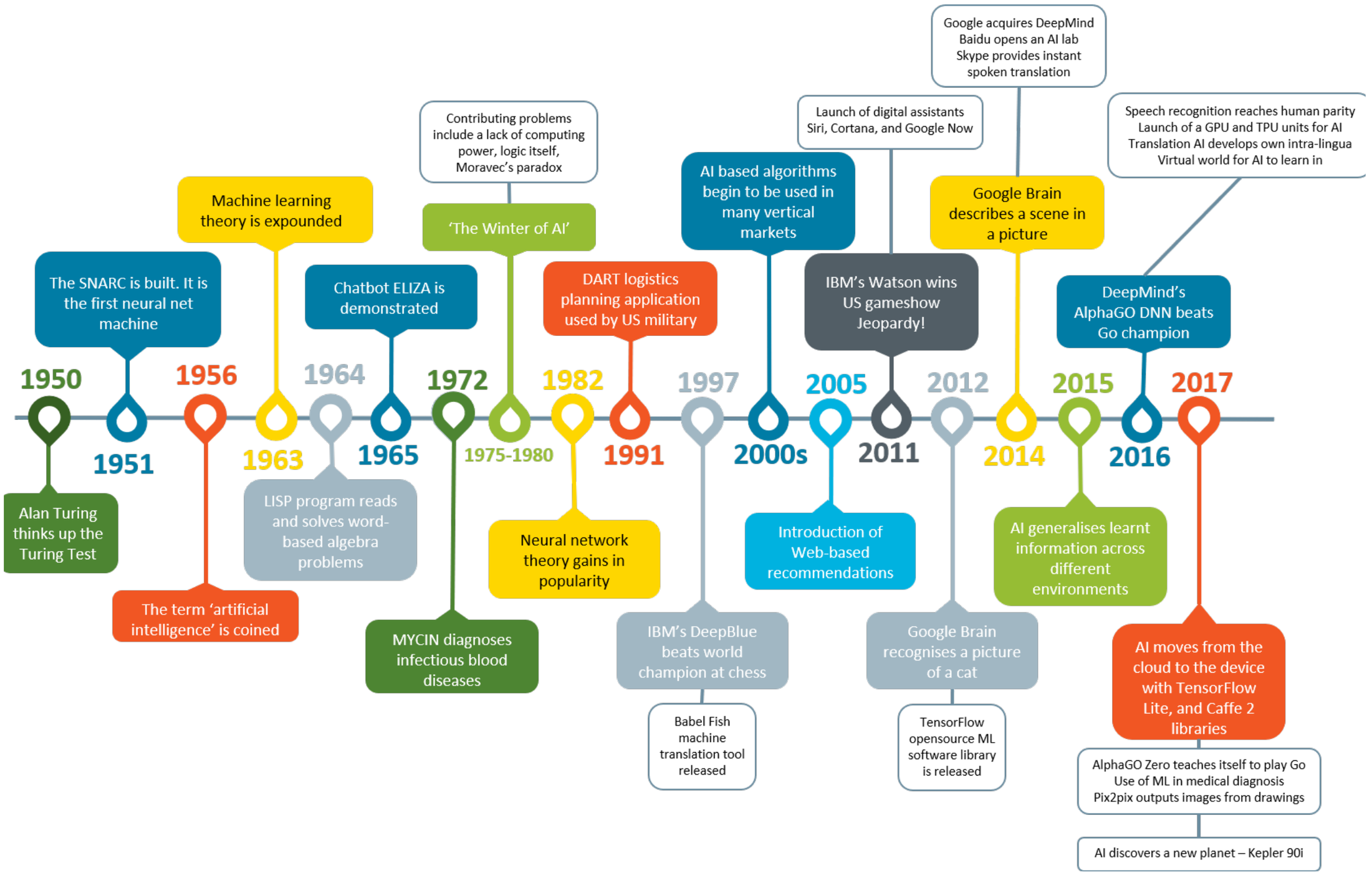
GROWTH OF COMPUTING

The accelerating pace of change and exponential growth in computing power will lead to the Singularity !

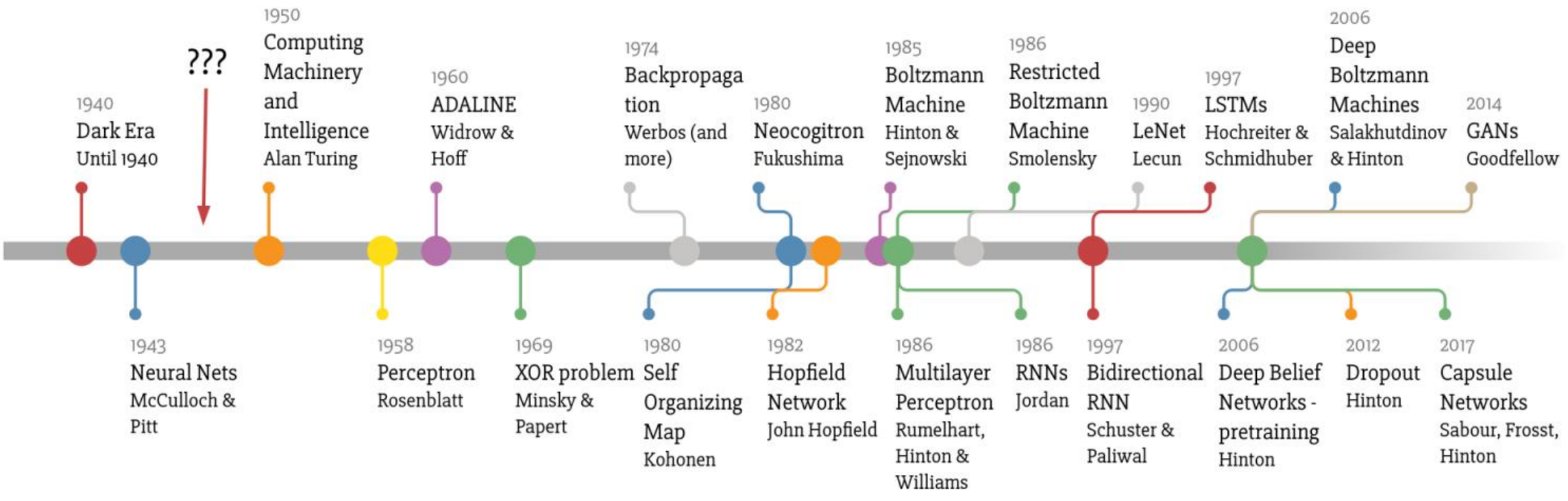


Reference : Lev Grossman (2011), 2045: The Year Man Becomes Immortal [TIME]

GROWTH OF AI


1950
Alan Turing thinks up the Turing Test
1951
The SNARC is built. It is the first neural net machine
1956
The term 'artificial intelligence' is coined
Machine learning theory is expounded
1963
1964
LISP program reads and solves word-based algebra problems
Chatbot ELIZA is demonstrated
1965
MYCIN diagnoses infectious blood diseases
1972
Contributing problems include a lack of computing power, logic itself, Moravec's paradox
'The Winter of AI'
1975-1980
1982
Neural network theory gains in popularity
DART logistics planning application used by US military
1991
1997
IBM's DeepBlue beats world champion at chess
Babel Fish machine translation tool released
2000s
AI based algorithms begin to be used in many vertical markets
2005
Introduction of Web-based recommendations
2011
IBM's Watson wins US gameshow Jeopardy!
Launch of digital assistants Siri, Cortana, and Google Now
2012
Google Brain recognises a picture of a cat
TensorFlow opensource ML software library is released
Google acquires DeepMind
Baidu opens an AI lab
Skype provides instant spoken translation
2014
Google Brain describes a scene in a picture
2015
AI generalises learnt information across different environments
2016
DeepMind's AlphaGO DNN beats Go champion
Speech recognition reaches human parity
Launch of a GPU and TPU units for AI
Translation AI develops own intra-lingua
Virtual world for AI to learn in
2017
AI moves from the cloud to the device with TensorFlow Lite, and Caffe 2 libraries
AlphaGO Zero teaches itself to play Go
Use of ML in medical diagnosis
Pix2pix outputs images from drawings
AI discovers a new planet – Kepler 90i

Deep Learning Timeline



Made by Favio Vázquez

MACHINE LEARNING

- Algorithms - Neural Networks, LSTM, Attention, BERT, Gan,….
- Tools/Frameworks - Tensorflow, Torch, MxNet,…
- Applications - Text, Speech, Audio, Video,….

Industries

Academics	Government	Language
Retail	Energy	Transport & Travel
Finance	Healthcare	Research
Manufacturing	Robotics	Software
Media	Recommendation engines	

Machine Learning in Software Development

Components of machine learning on source code

Machine Learning

Software Engineering

Programming Languages

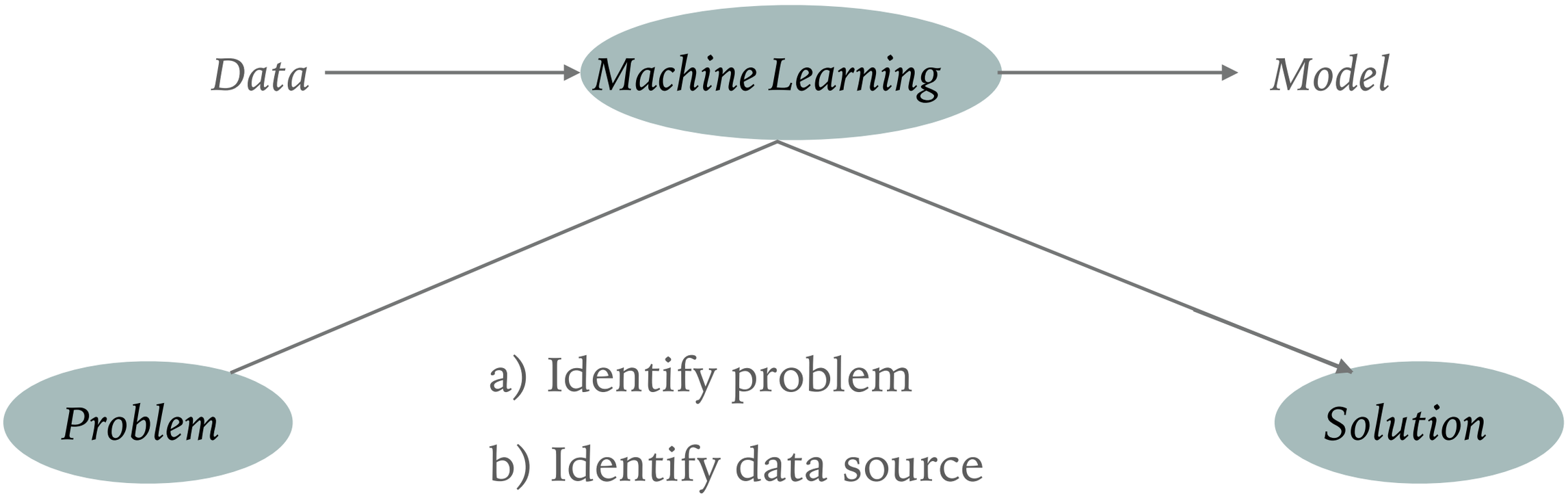
Natural Languages

GITHUB DATA

- Total repos : 100 M +
- Public repos : 28M+
- Total developers : 40 M+
- Pull requests : 87 M +
- Organisations : 2.7 M +
- Fortune 50 (open source): 27
- Number of files : 1 B +

Reference : https://octoverse.github.com/

INTRODCUTION



a) Identify problem
b) Identify data source
c) Data acquisitions and cleaning
d) Feature engineering
e) Modeling
f) Testing/validation
g) Deployment

PROBLEM : STATIC CODE ANALYSIS

"Static code analysis or static analysis is a method to debug a code by only examining the code and (no execution)"

- *Type checking*
- *Style checking*
- *Program understanding*
- *Program verification*
- *Property checking*
- *Bug Finding*
- *Bug finding*
- *Security review*
- *Design*
- *Auto complete*
- *Patch generation*
- *Risk analysis*
- *Recommendation*
- *Code search*
- *Clone detection*

EXAMPLE

Consider the following C function that prints a message to a specified file descriptor without performing any error checking:

```
void printMsg(FILE* file, char* msg) {
  fprintf(file, msg);
}
```

If either argument to this function is null, the program will crash. Programming defensively, we might check to make sure that both input parameters are non-null before printing the message, as follows:

```
void printMsg(FILE* file, char* msg) {
  if (file == NULL) {
    logError("attempt to print message to null file");
  } else if (msg == NULL) {
    logError("attempt to print null message");
  } else {
    fprintf(file, msg);
  }
}
```

Referenece : B rian Chess and J acob West (2007), Secure programming with static analysis

ANATOMY OF A SOFTWARE REPOSITORY

- Files + databases
- Source code [high level programming languages]
- Text [Natural languages] - comments, doc strings, README …
- Development history - when ? what ? who ? why ?
- Metadata - author, stars, owner…
- Issues
- …

torvalds / linux

Watch 6.8k | Star 84.6k | Fork 29.6k

Code | Pull requests 315 | Actions | Projects 0 | Security | Insights

Linux kernel source tree

887,859 commits | 1 branch | 0 packages | 633 releases | ∞ contributors | View license

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

torvalds Linux 5.5-rc4 — Latest commit fd69884 yesterday

Documentation	kunit: Rename 'kunitconfig' to '.kunitconfig'	8 days ago
LICENSES	LICENSES: Rename other to deprecated	8 months ago
arch	riscv: export flush_icache_all to modules	3 days ago
block	compat_ioctl: block: handle Persistent Reservations	10 days ago
certs	certs: Add wrapper function to check blacklisted binary hash	2 months ago
crypto	Merge tag 'tpmdd-next-20191219' of git://git.infradead.org/users/jjs/...	12 days ago
drivers	Merge tag 'scsi-fixes' of git://git.kernel.org/pub/scm/linux/kernel/g...	3 days ago
fs	Merge tag 'locks-v5.5-1' of git://git.kernel.org/pub/scm/linux/kernel...	2 days ago
include	ata: libahci_platform: Export again ahci_platform_<en/dis>able_phys()	5 days ago
init	early init: fix error handling when opening /dev/console	14 days ago
ipc	treewide: Use sizeof_field() macro	22 days ago

SOURCE CODE

- Two channels communication - programmers, computers.
- Compilable.
- Not just a linear sequence of tokens - non-linear (conditional jumps,..).
- Unambiguous.
- Infinite vocabulary.
- Much deeper (AST) trees than (parse) trees in text.

NATURAL LANGUAGE MODELS

- N-Gram models : These are generative models which define probability of creating a new sentence.
- The probability of the occurrence of a sequence w1,w2,w3.,…., wN in any data can be written as:

$$p(w_1, w_2, w_3, \ldots, w_m) = \prod_{i=l}^{m} P(w_i \mid w_1^{i-1}) \text{ where}$$

$$P(w_2 \mid w_1) = \frac{C(w_1, w_2)}{C(w_1)} \text{ and 'C' is a cooccurrence matrix.}$$

- Chain rule :

$$P(1,2,3) = P(1 \mid 2,3)P(2 \mid 3)P(3)$$

Reference : https://web.stanford.edu/~jurafsky/slp3/3.pdf

DISSECTING A CODE

- Lexical Analysis - code into a set of tokens using regex

```
if (ret) // probably true
  mat[x][y] = END_VAL;
```

This code produces the following sequence of tokens:

```
IF LPAREN ID(ret) RPAREN ID(mat) LBRACKET ID(x) RBRACKET LBRACKET
ID(y) RBRACKET EQUAL ID(END_VAL) SEMI
```

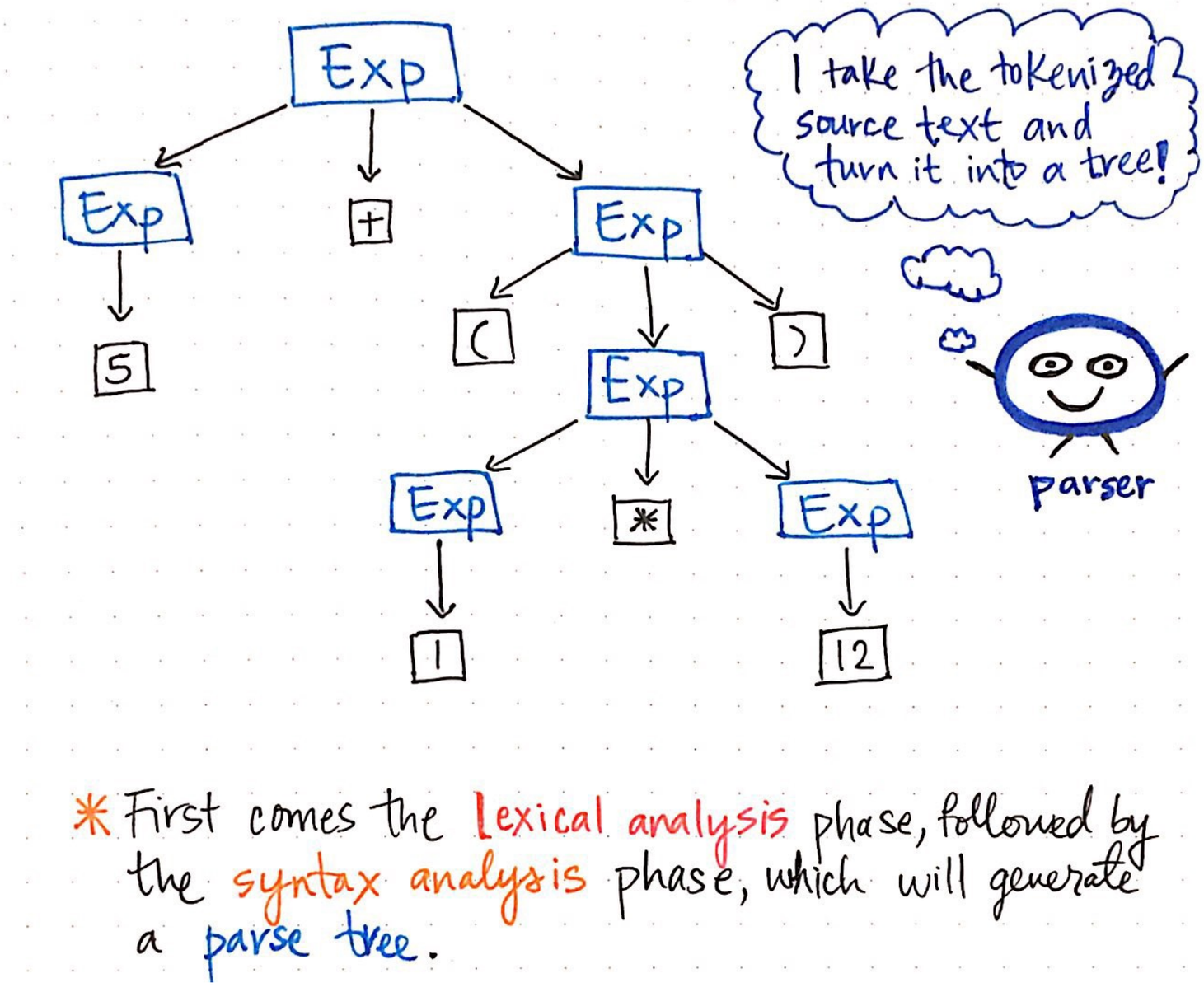
Lexical analysis

- Parsing - Using the tokens get a parse tree following some context free grammar (CFG)
- Abstract Syntax tree (AST) - Simplify parse tree
- Semantic analysis - type checking and symbol resolution
- Intermediate representation by compiler using AST

PARSE TREE

Expression:

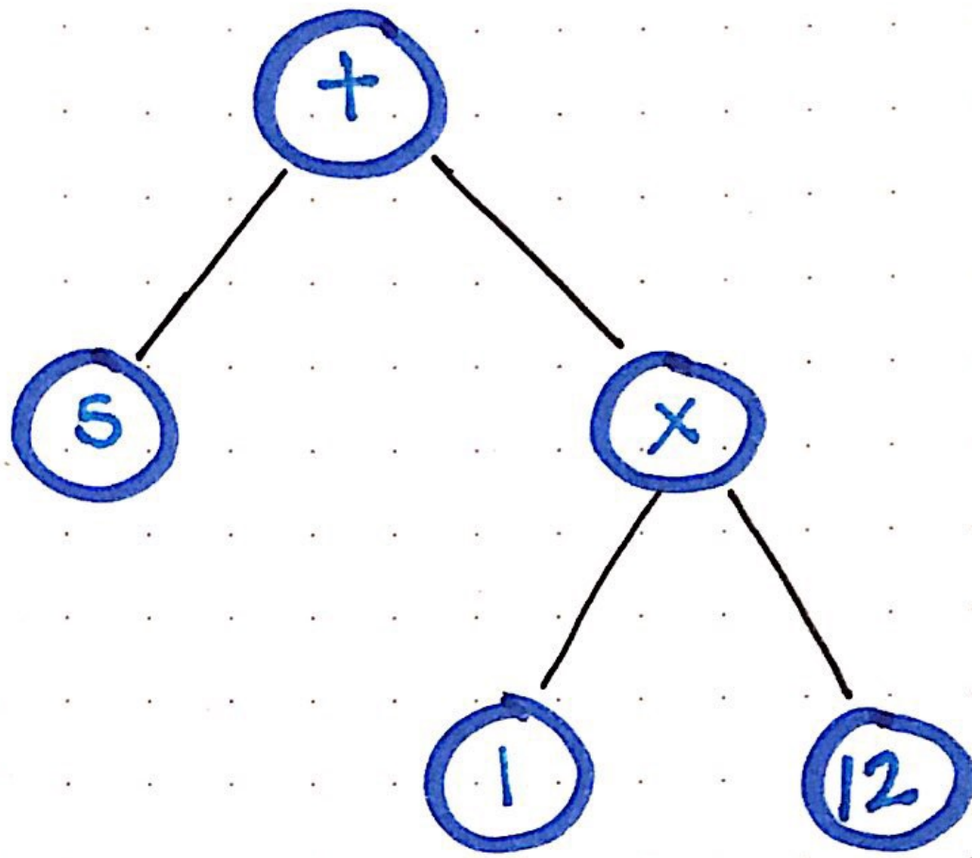
*5+(1 * 12)*



Reference : https://medium.com/basecs/leveling-up-ones-parsing-game-with-asts-d7a6fc2400ff

ABSTRACT SYNTAX TREE

An AST represents the exact same expression as our parse tree, but abstracts away from the concrete syntax.



Reference : https://medium.com/basecs/leveling-up-ones-parsing-game-with-asts-d7a6fc2400ff

ABSTRACT SYNTAX TREE

```
1 + 2
```

This could be represented in AST like this:

```
+ BinaryExpression
 - type: +
 - left_value:
  LiteralExpr:
   value: 1
 - right_vaue:
  LiteralExpr:
   value: 2
```

ABSTRACT SYNTAX TREE

```
if(2 > 6) {
    var d  = 90
    console.log(d)
}


IfStatement
 - condition
  + BinaryExpression
   - type: >
   - left_value: 2
   - right_value: 6
 - body
  [
    - Assign
        - left: 'd';
        - right:
            LiteralExpr:
            - value: 90
    - MethodCall:
         - instanceName: console
         - methodName: log
         - args: [


         ]
  ]
```

AST

```
class IfStmt {
    constructor(condition, body) {
        this.condition = condition
        this.body = body
    }
}
```

Now let's represent the below in the IfStmt class

```
if(9 > 7) {
    log('Yay!!')
}
```

The condition is a Binary operation, which will be represented like this:

```
const cond = new Binary(new Literal(9), "GREATER", new Literal(7))
```

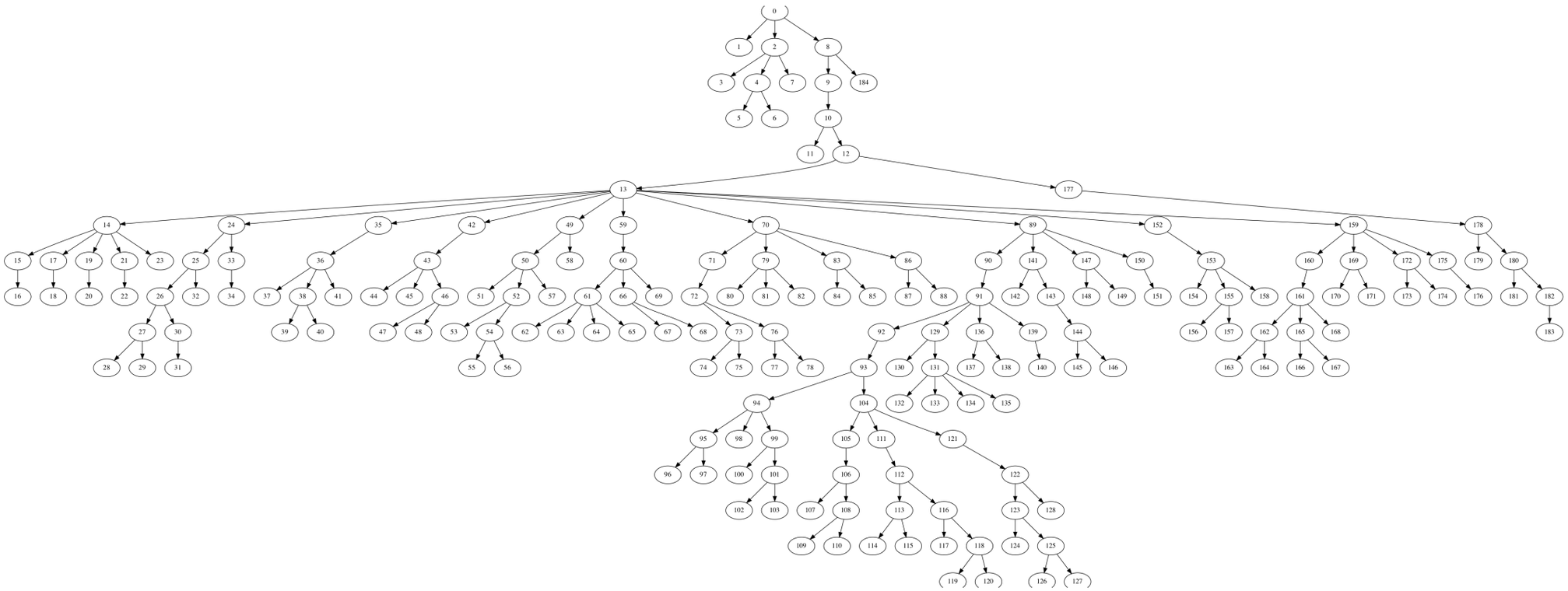
Reference : https://blog.bitsrc.io/what-is-an-abstract-syntax-tree-7502b71bde27

EXAMPLE

Bubble sort

```
import java.util.Scanner;
 class BubbleSort {
  public static void main(String []args) {
    int n, c, d, swap;
    Scanner in = new Scanner(System.in);
    System.out.println("Input number of integers to sort");
    n = in.nextInt();
    int array[] = new int[n];
    System.out.println("Enter " + n + " integers");
    for (c = 0; c < n; c++)
      array[c] = in.nextInt();
    for (c = 0; c < ( n - 1 ); c++) {
      for (d = 0; d < n - c - 1; d++) {
        if (array[d] > array[d+1]) /* For descending order use < */
        {
          swap       = array[d];
          array[d]   = array[d+1];
          array[d+1] = swap;
        }
      }
    }
    System.out.println("Sorted list of numbers");
    for (c = 0; c < n; c++)
      System.out.println(array[c]);
  }
}
```

EXAMPLE



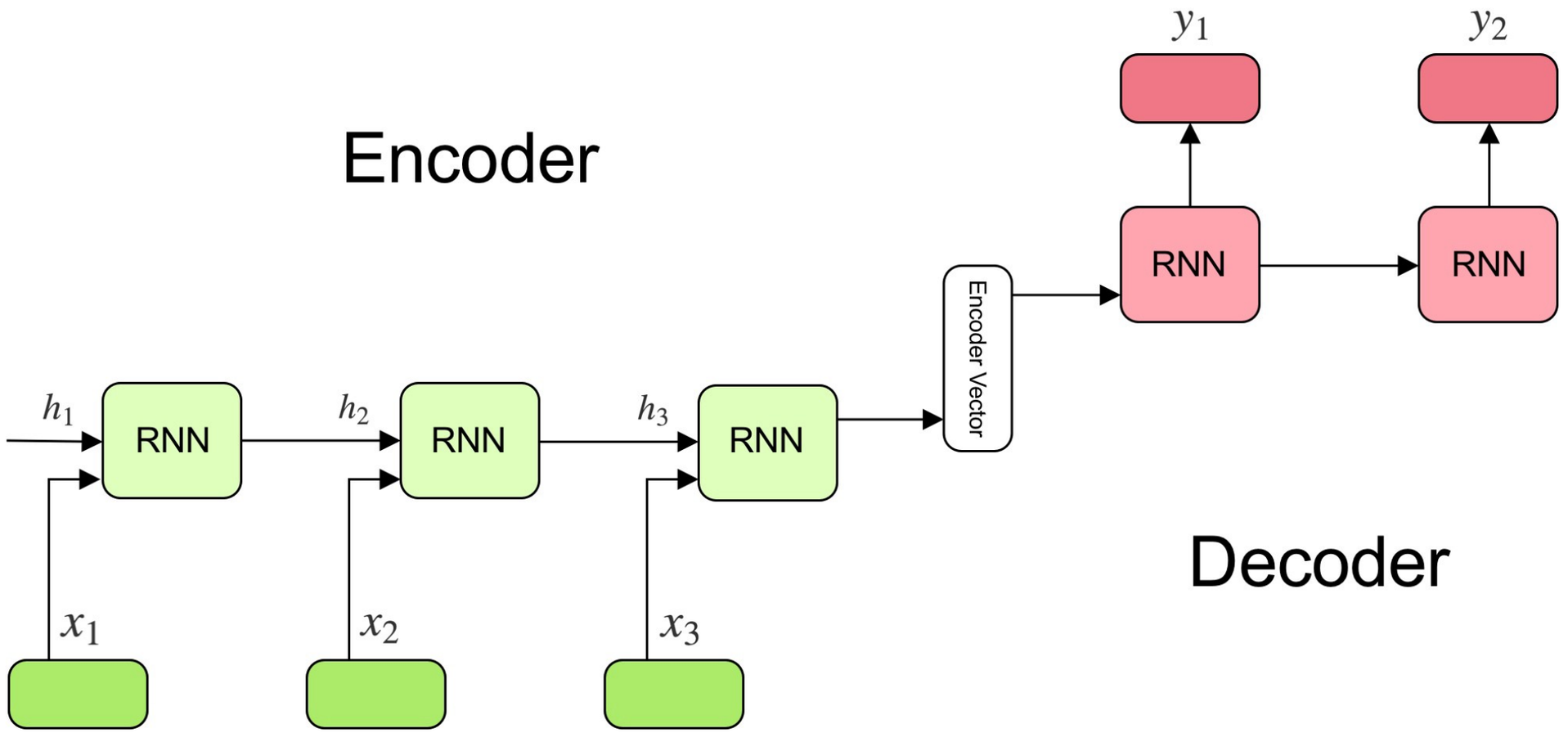
Anytree representation [https://anytree.readthedocs.io/en/latest/]

REVIEW

- A survey of the efforts in the direction of applying machine learning on source (big) code is given in [Allamanis et. el. (2018)].
- Like in languages, probabilistic models can be used to generate code, which can be in the form of a sequence of tokens or trees (AST).
- Some of the applications of code generation model are code completion, text to code, code synthesis, Bug detection, obfuscation, code review, information, extraction, syntax error correction, code search etc.
- Some of common models have been n-gram, RNN (LSTM) etc.

SEQ2SEQ MODELS

- In 2014 a team of google's researchers [Sutskever et. al. (2014)] proposed a sequence to sequence model based on encoder-decoder architecture for language translation.
- Since 2014 many applications have been found of seq2seq and many variations have been proposed.
- One of the attractive features of this model is that any arbitrary size sequence can be expressed in terms of a fixed size vector.
- The encoder-decoder model used LSTM units to remember the contexts.



Encoder-Decoder Model

ENCODER-DECODER MODEL

Let us consider we have an input sequence $x_1, x_2, \ldots, x_T$ and output sequence for every time step we can compute the internal state as:

$h_t = f(x_t, h_{t-1})$ ⟶ *Encoder*

We can represent the set of internal states as:

$c = q(h_1, h_2, h_3, \ldots, h_T)$

Now we can predict output sequence with probability :

$$P(y_1, y_2, \ldots, y_{T-1}) = \prod_{t-1}^{t=T} P(y_t | y_1, y_2, .., y_{t-1}, c) \longrightarrow \text{Decoder}$$

Seq2seq model considers $c = h_T$, however, attention models remove that limitation [Bahdanau et. Al (2016)].

SEQ2SEQ MODELS

- **Seq2Seq** models can be called baseline models and there are some important variations which are worths looking [Bahdanau, Cho & Bengio (2016)].
- The **attention transformer** model [Vasvani et. al. (2017)] claims superior performance as compared to seq2seq models.
- Further improvement has been claimed by a model named Bidirectional Encoder Representations from Transformers (**BERT**) as compared to transformer [Devlin (2018)].
- There is a model called **XLnet** [Yang (2019)] which claims better performance as compared to BERT.
- Most of the above models have been used in Language tasks.

REVIEW : TREELSTM

- One of the most important variations of the sequence to sequence model has been the TreeLST [Tai et. al. (2015)] which can be used for Tree like structures.
- Unlike a linear sequence of tokens nodes of a tree can have multiple children.
- TreeLSTM has been used to model sentences in NLP as well as source code ASTS.
- You can find a simplified version of TreeLSTM on my Github page page.

TREELSTM

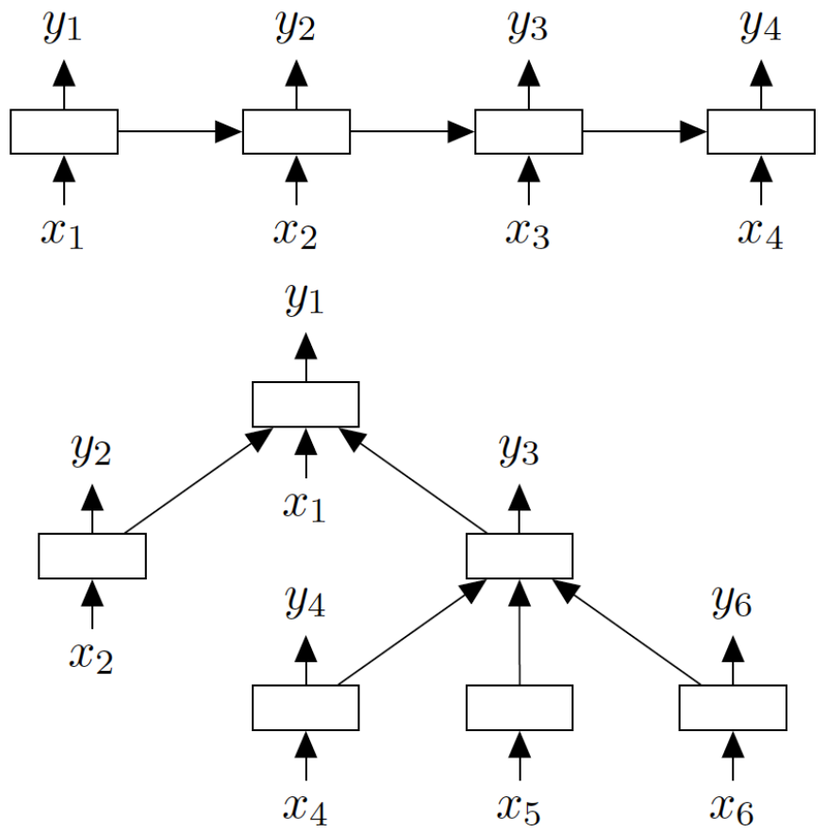


Figure 1: **Top:** A chain-structured LSTM network. **Bottom:** A tree-structured LSTM network with arbitrary branching factor.

Reference : Tai et. al. (2015)

REVIEW : CODE2SEQ & CODE2VEC

- One of the interesting uses cases for applying machine learning on source code could be generating text from code which can be in the form of variables, class or methods names or doc string or code summarisation.
- Using the methods or Neural Machine Translation (NMT) models based on Long-Short-Memory-Model (LSTM) [Alon & Levy (2018a, 2018b)] proposed an approach which can be used for code summarisation and code captioning.
- The above approach represents code as set of paths of paths in the abstract syntax tree.

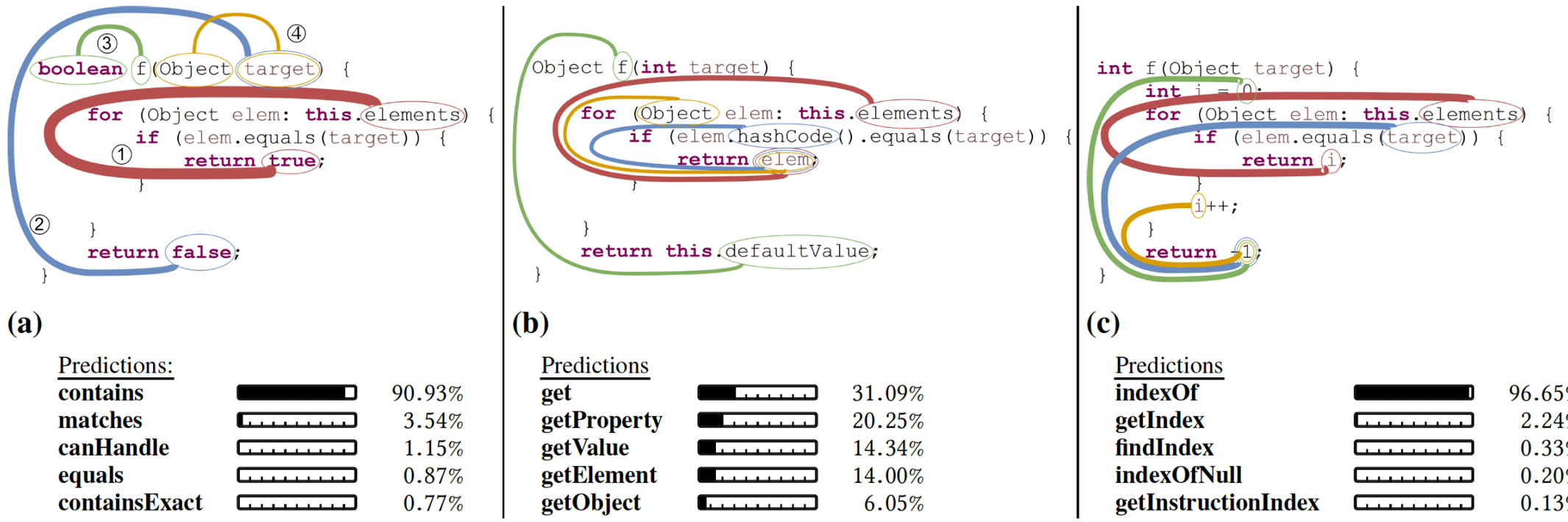


Fig. 2. An example for three methods that have a similar syntactic structure, but our model successfully captures the delicate differences between them and manages to predict meaningful names. The widths of the colored paths are proportional to the attention that each path was given.

Reference : Alon et. al. (2018): code2vec

TREE2TREE

- Inspired from seq2seq model tree2tree models [Chen et. al. (2018), Chakravarti et. al. (2018, 2019)] have been also proposed.
- Chakravarti et. al. work is a part of code change suggestion engine CODIT.

Edit Sizes	#Examples	Top 1 CODIT	Top 1 Seq2Seq	Top 2 CODIT	Top 2 Seq2Seq	Top 5 CODIT	Top 5 Seq2Seq	Top 10 CODIT	Top 10 Seq2Seq	Top 20 CODIT	Top 20 Seq2Seq
1	3368	60	5	101	15	124	64	138	76	170	94
2 - 5	2061	9	2	21	4	60	6	73	12	95	21
6 - 10	1306	156	159	156	176	160	177	161	179	161	185
Total	6735	225 3.34%	166 2.46%	278 4.13%	195 2.90%	344 5.11%	247 3.67%	372 5.52%	267 3.96%	426 6.33%	300 4.45%
Gain $(\frac{CODIT-Seq2Seq}{Seq2Seq})$		35.77%		42.41%		39.24%		39.39%		42.25%	

TABLE VI: **Performance of CODIT suggesting concrete patches**

SUMMARY & CONCLUSIONS

- In this part we have introduce the problems of machine learning on source code.
- A framework based on neural network (RNN/LSTM) was introduced.
- Some applications for a static code analysis were discussed.
- Key references and contribution were highlighted.
- In the next part we will discuss some more theoretical background and a concrete implementation.
- Tools, techniques and coding will be also be discussed.

REFERENCES

- Allamanis et. al. (2018), ACM Computing Surveys, *A Survey of Machine Learning for Big Code and Naturalness*
- Alon et. al. (2018), ICLR'19, *code2seq: Generating Sequences from Structured Representations of Code*
- Alon et. al. (2018), POPL 2019, *code2vec: Learning Distributed Representations of Code*
- Sutskever et. al. (2015), *Sequence to Sequence Learning with Neural Networks*
- Tai et. al. (2015), *ACL 2015*, *Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks*

REFERENCES

- Brain Chess & Jacob West (2007), *Secure Programming with Static Analysis* [Addison-Wesley Professional].
- Chen et. al. (2018), *Tree-to-tree Neural Networks for Program Translation*, [arXiv:1802.03691].
- Chakravarti et. al. (2018), *Tree2Tree Neural Translation Model for LearningSource Code Changes*, [arXiv:1810.00314].
- Chakravarti et. al. (2019), *CODIT: Code Editing with Tree-Based Neural Machine Translation*, [arXiv:1810.00314].
- Bahdanau, Cho & Bengio (2016), *Neural Machine Translation by jointly learning to align & translate*, [arXiv:1409.0473].

REFERENCES

- Vaswani et. al. (2017), *Attention is all what you need,* [arXiv:1706.03762].
- Devlin et. al. (2018), *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,* [arXiv:1810.04805].
- Yang et. al. (2019), XLNet: Generalized Autoregressive Pretraining for Language Understanding, [arXiv:1906.08237].

ASSIGNMENTS

- Read the survey paper of machine learning on source code.
- Set up python 3.6 anaconda environment & try out keras seq2seq examples.
- Read about Abstract Syntax Tree (AST)
- Install & setup Bebelefist AST server.
- Clone the ‘Big Code’ from my GitHub repo and have a look at that.

THANK YOU