From Deep Learning to Human Judgments: Lessons for Genetic Improvement

Westley Weimer
University of Michigan

(11th International Workshop on Genetic Improvement, 9 July 2022)

Outline (45+15 minutes)

- An Existential Crisis?
- Summary of Recent Advances
  - Generative Pre-Trained Transformers
- Concerns
  - Cost
  - Novelty
  - Problem Statement
  - Evaluations
- Recommendations:
  - Deception, Eyes, Algorithms, etc.
- Industrial Deployments
- Summary

This talk will provide a gentle introduction to these topics

We will benefit from a vigorous discussion!

Many of you may be familiar with other aspects of these issues

Program Improvement, AI and Machine Learning

- Increasing use of techniques associated with AI and ML (e.g., neural networks, language models, machine translation approaches, etc.) for program repair and improvement
- Researchers from other backgrounds (e.g., EC, SE, PL) have expressed significant concerns
  - Heard from PC members, collaborators and non-collaborators, multiple countries, etc.

- Example: "They will descend like a plague of locusts, convince everyone it is another problem defeated by their hammer, and then move on."

Fear, Uncertainty, and Doubt?

- Important to separate out reactionary resistance to change vs. more nuanced critiques
  - If these techniques really do entirely solve this problem, excellent!
  - But do they *entirely* solve *this* problem?

- Common critiques
  - Problem formulation: assuming perfect fault localization
  - Assessment and evaluation: internal metrics
  - Foundational limitations: lack of novel synthesis
  - Moral accessibility concerns: monetary cost of training models excludes participation

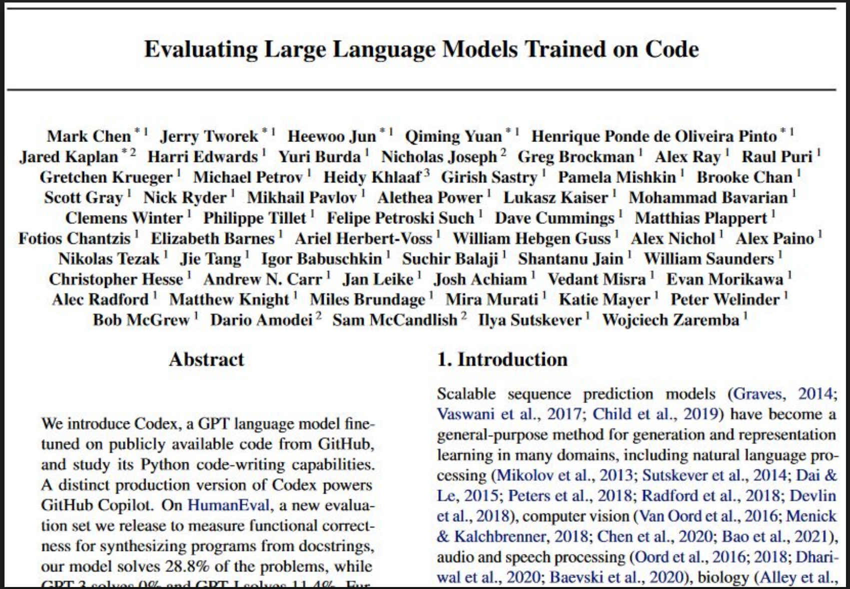
Challenge and Opportunity

“The rise of language models raises many interesting connections [...] At the most basic or unit level, there is a dire need to improve the code generated by language models like Codex [...] a need to understand the kind of semantic errors that lurk in such auto-generated code [...] value in proposing analysis or fixing mechanisms specifically for auto-generated code [...] However, there is the opportunity to expand on these prompts to capture the power of program synthesis. Program synthesis, or programming by example approaches, differ from language model-based approaches primarily in the ability to synthesize code which was never seen before.”

- Abhik Roychoudhury, NUS (SemFix, Angelix, Concolic Program Repair, etc.)

Deep Learning & Language

- OpenAI Codex is a Generative Pre-trained Transformer (GPT) approach in which a neural network based on a deep learning model is trained on an enormous corpus of text
- It can produce prose with human-equivalent fluency



Evaluating Large Language Models Trained on Code

Mark Chen *[1] Jerry Tworek *[1] Heewoo Jun *[1] Qiming Yuan *[1] Henrique Ponde de Oliveira Pinto *[1] Jared Kaplan *[2] Harri Edwards [1] Yuri Burda [1] Nicholas Joseph [2] Greg Brockman [1] Alex Ray [1] Raul Puri [1] Gretchen Krueger [1] Michael Petrov [1] Heidy Khlaaf [3] Girish Sastry [1] Pamela Mishkin [1] Brooke Chan [1] Scott Gray [1] Nick Ryder [1] Mikhail Pavlov [1] Alethea Power [1] Lukasz Kaiser [1] Mohammad Bavarian [1] Clemens Winter [1] Philippe Tillet [1] Felipe Petroski Such [1] Dave Cummings [1] Matthias Plappert [1] Fotios Chantzis [1] Elizabeth Barnes [1] Ariel Herbert-Voss [1] William Hebgen Guss [1] Alex Nichol [1] Alex Paino [1] Nikolas Tezak [1] Jie Tang [1] Igor Babuschkin [1] Suchir Balaji [1] Shantanu Jain [1] William Saunders [1] Christopher Hesse [1] Andrew N. Carr [1] Jan Leike [1] Josh Achiam [1] Vedant Misra [1] Evan Morikawa [1] Alec Radford [1] Matthew Knight [1] Miles Brundage [1] Mira Murati [1] Katie Mayer [1] Peter Welinder [1] Bob McGrew [1] Dario Amodei [2] Sam McCandlish [2] Ilya Sutskever [1] Wojciech Zaremba [1]

Abstract

We introduce Codex, a GPT language model fine-tuned on publicly available code from GitHub, and study its Python code-writing capabilities. A distinct production version of Codex powers GitHub Copilot. On HumanEval, a new evaluation set we release to measure functional correctness for synthesizing programs from docstrings, our model solves 28.8% of the problems, while GPT-3 solves 0% and GPT-J solves 11.4%. Fur

1. Introduction

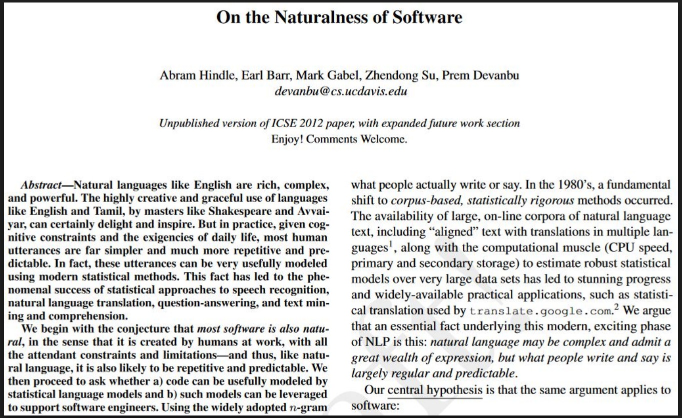
Scalable sequence prediction models (Graves, 2014; Vaswani et al., 2017; Child et al., 2019) have become a general-purpose method for generation and representation learning in many domains, including natural language processing (Mikolov et al., 2013; Sutskever et al., 2014; Dai & Le, 2015; Peters et al., 2018; Radford et al., 2018; Devlin et al., 2018), computer vision (Van Oord et al., 2016; Menick & Kalchbrenner, 2018; Chen et al., 2020; Bao et al., 2021), audio and speech processing (Oord et al., 2016; 2018; Dhariwal et al., 2020; Baevski et al., 2020), biology (Alley et al.,

A.I. Is Mastering Language. Should We Trust What It Says?

OpenAI's GPT-3 and other neural nets can now write original prose with mind-boggling fluency — a development that could have profound implications for the future.

GitHub Copilot

- Beyond natural language, models can be trained and applied to source code
- Using NLP on code at scale is not new, but the way it is playing out now is



On the Naturalness of Software

Abram Hindle, Earl Barr, Mark Gabel, Zhendong Su, Prem Devanbu
devanbu@cs.ucdavis.edu

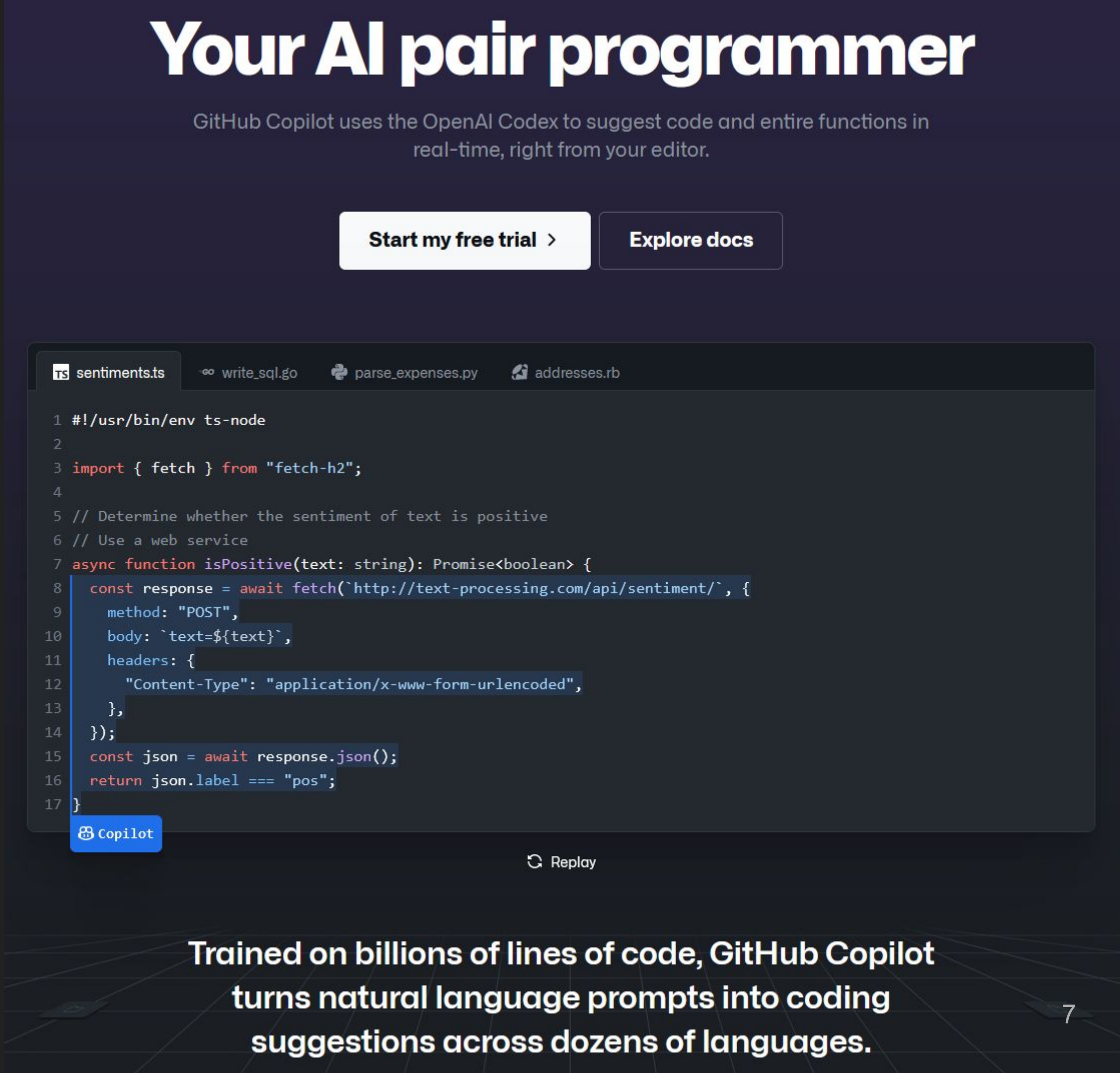
Unpublished version of ICSE 2012 paper, with expanded future work section
Enjoy! Comments Welcome.

***Abstract*—Natural languages like English are rich, complex, and powerful. The highly creative and graceful use of languages like English and Tamil, by masters like Shakespeare and Avvaiyar, can certainly delight and inspire. But in practice, given cognitive constraints and the exigencies of daily life, most human utterances are far simpler and much more repetitive and predictable. In fact, these utterances can be very usefully modeled using modern statistical methods. This fact has led to the phenomenal success of statistical approaches to speech recognition, natural language translation, question-answering, and text mining and comprehension.**

We begin with the conjecture that *most software is also natural*, in the sense that it is created by humans at work, with all the attendant constraints and limitations—and thus, like natural language, it is also likely to be repetitive and predictable. We then proceed to ask whether a) code can be usefully modeled by statistical language models and b) such models can be leveraged to support software engineers. Using the widely adopted n-gram

what people actually write or say. In the 1980's, a fundamental shift to *corpus-based, statistically rigorous* methods occurred. The availability of large, on-line corpora of natural language text, including "aligned" text with translations in multiple languages[1], along with the computational muscle (CPU speed, primary and secondary storage) to estimate robust statistical models over very large data sets has led to stunning progress and widely-available practical applications, such as statistical translation used by `translate.google.com`.[2] We argue that an essential fact underlying this modern, exciting phase of NLP is this: *natural language may be complex and admit a great wealth of expression, but what people write and say is largely regular and predictable.*

Our central hypothesis is that the same argument applies to software:



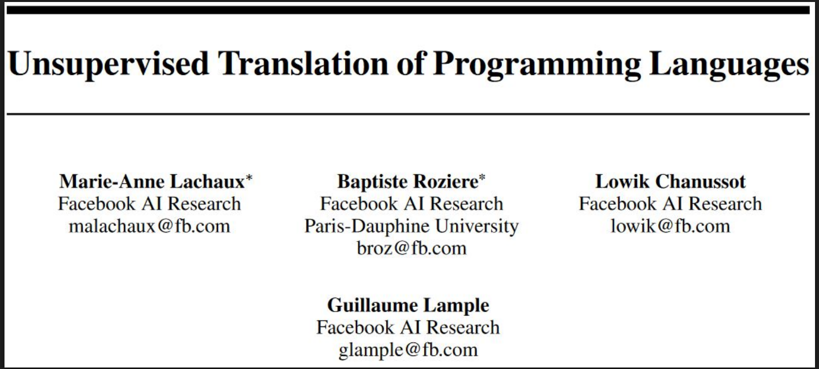
Your AI pair programmer
GitHub Copilot uses the OpenAI Codex to suggest code and entire functions in real-time, right from your editor.
Start my free trial >
Explore docs
sentiments.ts
write_sql.go
parse_expenses.py
addresses.rb

```
#!/usr/bin/env ts-node

import { fetch } from "fetch-h2";

// Determine whether the sentiment of text is positive
// Use a web service
async function isPositive(text: string): Promise<boolean> {
  const response = await fetch(`http://text-processing.com/api/sentiment/`, {
    method: "POST",
    body: `text=${text}`,
    headers: {
      "Content-Type": "application/x-www-form-urlencoded",
    },
  });
  const json = await response.json();
  return json.label === "pos";
}
```

Trained on billions of lines of code, GitHub Copilot turns natural language prompts into coding suggestions across dozens of languages.

Facebook TransCoder

- The TransCoder technique from Facebook AI Research uses a transformer (encoder-decoder) architecture to translate source code between languages
- It predates the emphasis on Codex, and its specific emphasis on readability (and evaluations) makes it relevant for a Genetic Improvement discussion



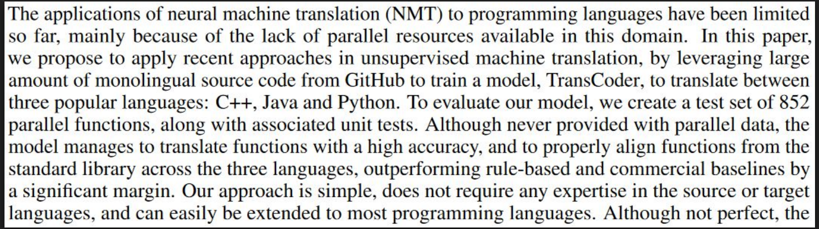
Unsupervised Translation of Programming Languages

Marie-Anne Lachaux*
Facebook AI Research
malachaux@fb.com

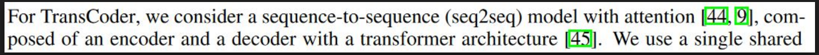
Baptiste Roziere*
Facebook AI Research
Paris-Dauphine University
broz@fb.com

Lowik Chanussot
Facebook AI Research
lowik@fb.com

Guillaume Lample
Facebook AI Research
glample@fb.com



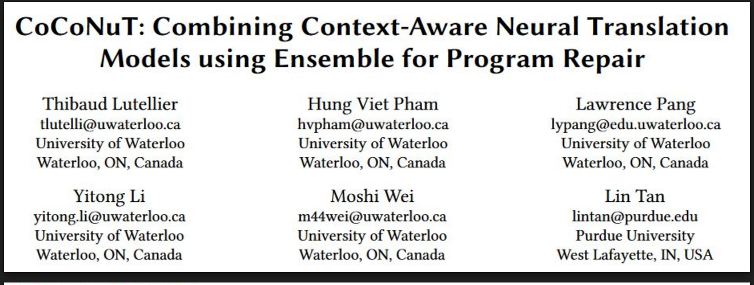
The applications of neural machine translation (NMT) to programming languages have been limited so far, mainly because of the lack of parallel resources available in this domain. In this paper, we propose to apply recent approaches in unsupervised machine translation, by leveraging large amount of monolingual source code from GitHub to train a model, TransCoder, to translate between three popular languages: C++, Java and Python. To evaluate our model, we create a test set of 852 parallel functions, along with associated unit tests. Although never provided with parallel data, the model manages to translate functions with a high accuracy, and to properly align functions from the standard library across the three languages, outperforming rule-based and commercial baselines by a significant margin. Our approach is simple, does not require any expertise in the source or target languages, and can easily be extended to most programming languages. Although not perfect, the



For TransCoder, we consider a sequence-to-sequence (seq2seq) model with attention [44, 9], composed of an encoder and a decoder with a transformer architecture [45]. We use a single shared

Example in Program Repair: CoCoNuT

- We now have all of the building blocks to apply directly to APR
- The popular CoCoNuT project uses AI and deep learning and views program repair as translating from buggy to correct source code
- It reports very strong results, fixing 509 bugs (inc. 309 not fixed by 27 other baseline techniques) across 4 languages



CoCoNuT: Combining Context-Aware Neural Translation Models using Ensemble for Program Repair

Thibaud Lutellier
tlutelli@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

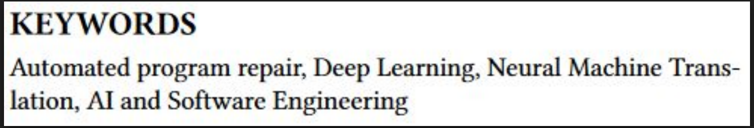
Hung Viet Pham
hvpham@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Lawrence Pang
lypang@edu.uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Yitong Li
yitong.li@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

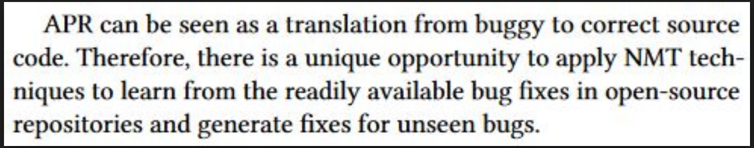
Moshi Wei
m44wei@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Lin Tan
lintan@purdue.edu
Purdue University
West Lafayette, IN, USA

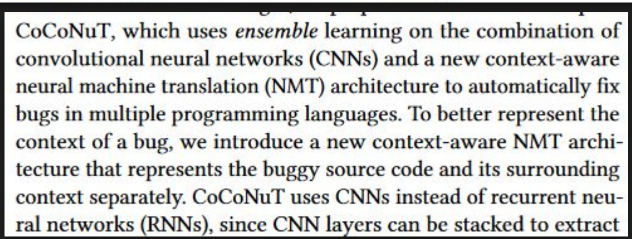


KEYWORDS

Automated program repair, Deep Learning, Neural Machine Translation, AI and Software Engineering



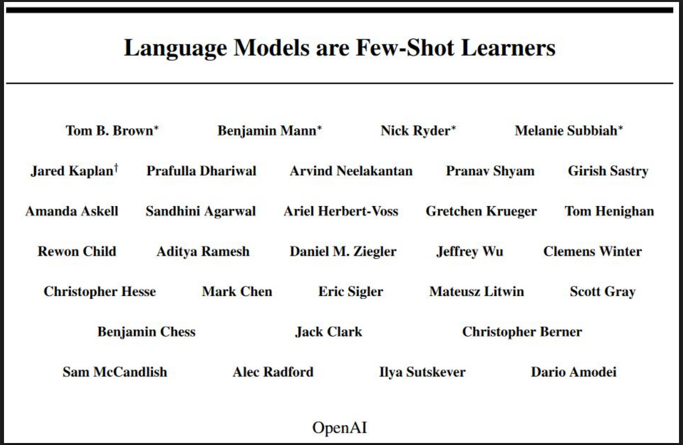
APR can be seen as a translation from buggy to correct source code. Therefore, there is a unique opportunity to apply NMT techniques to learn from the readily available bug fixes in open-source repositories and generate fixes for unseen bugs.



CoCoNuT, which uses *ensemble* learning on the combination of convolutional neural networks (CNNs) and a new context-aware neural machine translation (NMT) architecture to automatically fix bugs in multiple programming languages. To better represent the context of a bug, we introduce a new context-aware NMT architecture that represents the buggy source code and its surrounding context separately. CoCoNuT uses CNNs instead of recurrent neural networks (RNNs), since CNN layers can be stacked to extract

Training is Critical

- Training large models really matters, as the GPT-3 paper notes: “we show that scaling up language models greatly improves task-agnostic, few-shot performance, sometimes even reaching competitiveness with prior state-of-the-art fine tuning approaches”
  - That is, a model trained on a large-enough corpus matches or outperforms approaches specialized for specific tasks



Language Models are Few-Shot Learners

Tom B. Brown* Benjamin Mann* Nick Ryder* Melanie Subbiah*
Jared Kaplan† Prafulla Dhariwal Arvind Neelakantan Pranav Shyam Girish Sastry
Amanda Askell Sandhini Agarwal Ariel Herbert-Voss Gretchen Krueger Tom Henighan
Rewon Child Aditya Ramesh Daniel M. Ziegler Jeffrey Wu Clemens Winter
Christopher Hesse Mark Chen Eric Sigler Mateusz Litwin Scott Gray
Benjamin Chess Jack Clark Christopher Berner
Sam McCandlish Alec Radford Ilya Sutskever Dario Amodei

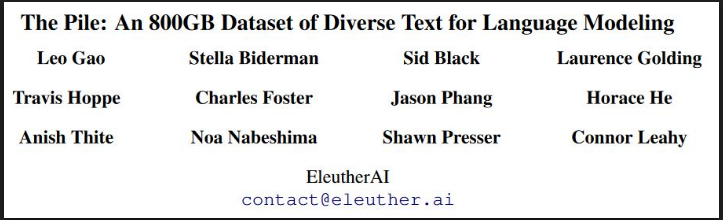
OpenAI

OpenAI Codex shows the limits of large language models

Ben Dickson
@BenDee983

July 18, 2021 12:15 PM

Training is Expensive


The Pile: An 800GB Dataset of Diverse Text for Language Modeling

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, Connor Leahy

EleutherAI
contact@eleuther.ai

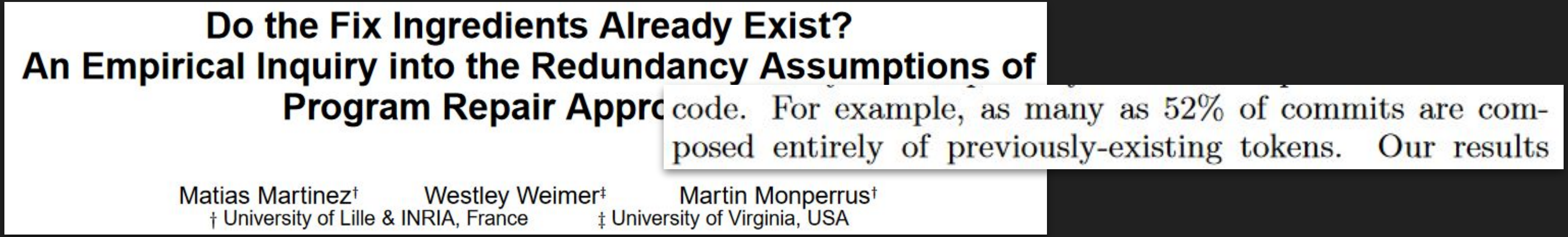
- GPT-3 was trained on 50x more than GPT-2 (600 GB) resulting in a 175 Billion parameter model. GPT-J was trained an 800 GB dataset. Copilot was trained on billions of lines of code.
- The Codex paper notes “First, Codex is not sample efficient to train [...] The original training of GPT-3-12B consumed hundreds of petaflop/s-days of compute, while fine-tuning it to create Codex-12B consumed a similar amount of compute. This training was performed on a platform (Azure) that purchases carbon credits [...]”
- Newer datasets (e.g., C4) are larger, with maintainers directly recommending the use of distributed cloud services for their use.

Training Concerns

- As a result, many researchers are morally concerned about the training costs (etc.) required for these techniques going forward
- Beyond environmental and “fairness in AI” concerns, my informal summary:
  - A generic model trained on a large corpus outperforms prior research
  - Training sizes have increased dramatically even within the last two years
  - Modern peer review *de facto* requires an X% improvement over the state of the art
  - Researchers need publications (e.g., for tenure or for students)
- Therefore, less-resourced researchers cannot afford to participate in fields dominated by such models
  - Both cannot afford the cloud computing training time
  - And also cannot afford to do “pure research” and then not get publications
- Overheard: “Soon only big companies will be able to participate.”

Novel Code Creation Concerns

- Approaches that generate based on pre-training are not suitable for creating new code not present in the training data
  - This is a nuanced claim, since they can rearrange trained words in different orders
  - GPT is good at "using novel words in a sentence after seeing them defined only once"
- By contrast
  - A semantics-based approach like SemFix or Angelix can create unseen code (e.g., by solving logical or arithmetic constraints)
  - A template-based approach may create unseen code via instantiation (but see "nuanced")
- The impact of this is uncertain (CoCoNuT success vs. ~50% upper bound)


Do the Fix Ingredients Already Exist? An Empirical Inquiry into the Redundancy Assumptions of Program Repair Appro

code. For example, as many as 52% of commits are composed entirely of previously-existing tokens. Our results

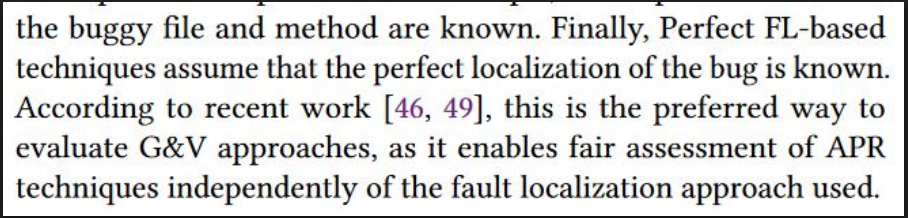
Matias Martinez† Westley Weimer‡ Martin Monperrus†
† University of Lille & INRIA, France ‡ University of Virginia, USA

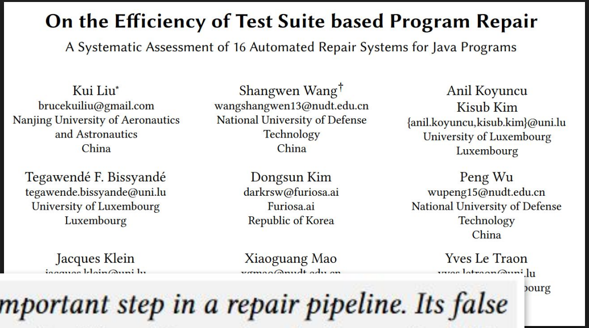
Problem Statement Concerns

- In NLP settings, the problem is often to produce the text that comes next
  - Given these X tokens, what should the next Y tokens be?
  - Others are possible (e.g., translate these X tokens from language A into language B)
- This can be cast naturally to program repair or improvement
- Informally: “Delete the buggy tokens, then given all of the previous tokens in the program before the bug, what new code should be placed there?”
- This formulation assumes perfect fault localization
  - In practice, fault localization is difficult in many contexts
    - Some security bugs (e.g., cross-site scripting or SQL code injection), some multi-threaded bugs, some entire domains (e.g., Verilog circuit designs), etc.

Fault Localization in Recent Evaluations

- The CoCoNuT paper, for example, describes using perfect fault localization to admit a fair comparison between generate-and-validate techniques
  - To me, that per se is quite reasonable
- The transitive argument is tricky
  - Ref [49] there is Liu et al.:
  - The paper calls out that it only applies to template-based tools and that constraint-based tools (e.g., ACS, Nopol) were not equally sensitive
  - Would GPT approaches be impacted more or less?


the buggy file and method are known. Finally, Perfect FL-based techniques assume that the perfect localization of the bug is known. According to recent work [46, 49], this is the preferred way to evaluate G&V approaches, as it enables fair assessment of APR techniques independently of the fault localization approach used.


On the Efficiency of Test Suite based Program Repair

A Systematic Assessment of 16 Automated Repair Systems for Java Programs

Kui Liu*
brucekuiliu@gmail.com
Nanjing University of Aeronautics and Astronautics
China

Shangwen Wang†
wangshangwen13@nudt.edu.cn
National University of Defense Technology
China

Anil Koyuncu
Kisub Kim
{anil.koyuncu,kisub.kim}@uni.lu
University of Luxembourg
Luxembourg

Tegawendé F. Bissyandé
tegawende.bissyande@uni.lu
University of Luxembourg
Luxembourg

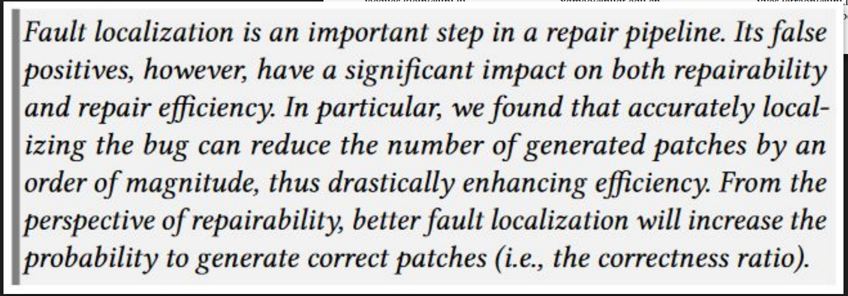
Dongsun Kim
darkrsw@furiosa.ai
Furiosa.ai
Republic of Korea

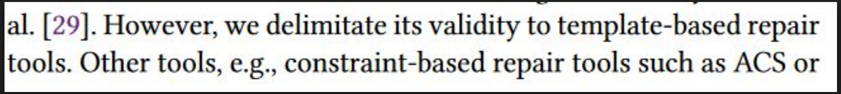
Peng Wu
wupeng15@nudt.edu.cn
National University of Defense Technology
China

Jacques Klein

Xiaoguang Mao

Yves Le Traon


Fault localization is an important step in a repair pipeline. Its false positives, however, have a significant impact on both repairability and repair efficiency. In particular, we found that accurately localizing the bug can reduce the number of generated patches by an order of magnitude, thus drastically enhancing efficiency. From the perspective of repairability, better fault localization will increase the probability to generate correct patches (i.e., the correctness ratio).

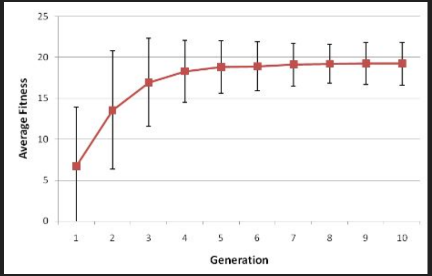

al. [29]. However, we delimitate its validity to template-based repair tools. Other tools, e.g., constraint-based repair tools such as ACS or

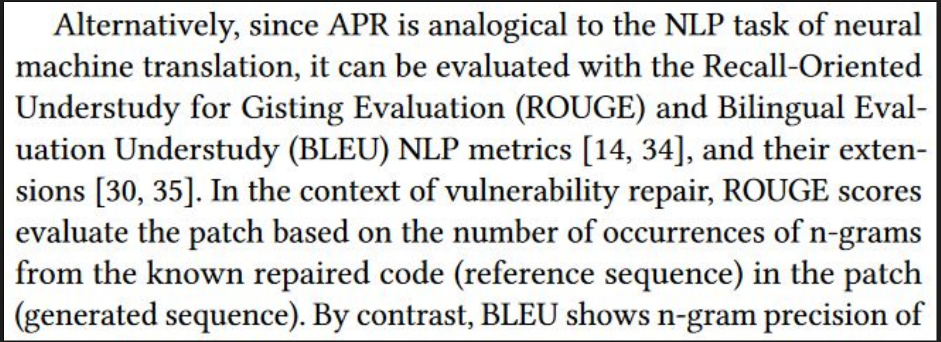
Evaluation Metrics

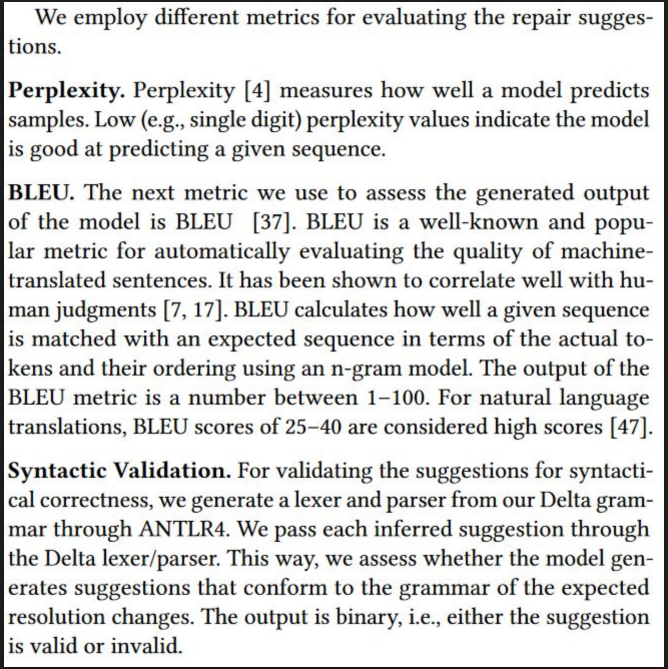
- In NLP domains, metrics such as ROUGE and BLEU and Perplexity are used
  - Recall-Oriented Understudy for Gisting Evaluation looks at the overlap of sequences of words between the reference and the output
  - BiLingual Evaluation Understudy uses sequence precision and brevity between reference sentences and output sentences
  - Perplexity measures how well a probability distribution predicts a sample, often in a "bits required per word" sense ("is this word common or expected here?")
  - Reference match measures perfectly matching the ground truth reference
- Metrics like ROUGE are syntactic, not semantic (e.g., do not handle synonyms or meaning)
  - Human1: "The cat is on the mat." Human2: "There is a cat on the mat."
  - Candidate3: "There is a cat on the mat." BLUE score is 7/7 = 1.0
  - Candidate4: "Mat the cat is on a there." BLUE score is 7/7 = 1.0

Appropriate Metric Selection

- To be clear: NLP metrics may be entirely appropriate in many situations
  - Comparing algorithmic advances between models
  - Researchers in another discipline first considering this problem domain
  - Elucidating internal algorithm behavior
- Just as we might measure “number of generations to produce a patch” as well as “number of programs improved”
  - An end user will care more about “number of programs improved”
  - But we, as researchers, may use information about a population search as a function of generation to guide internal decisions, study convergence, etc.
  - Example: early GenProg papers at GECCO did just that
  - Danger: “X uses fewer generations than Y so X is better than Y”
- Examples are illustrative of popularity, not “call outs”



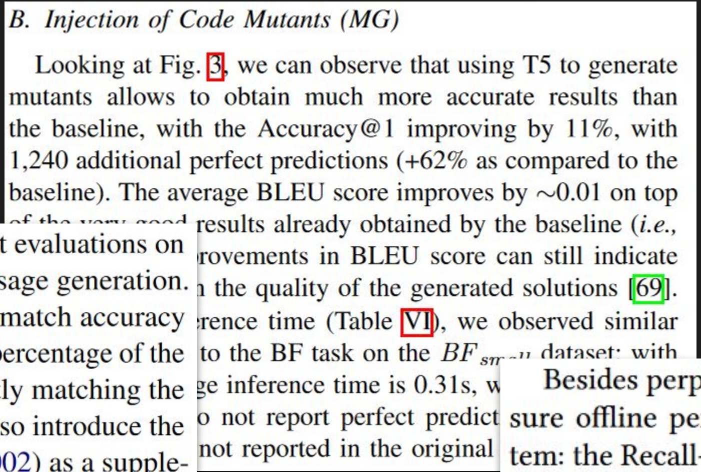

Alternatively, since APR is analogical to the NLP task of neural machine translation, it can be evaluated with the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) and Bilingual Evaluation Understudy (BLEU) NLP metrics [14, 34], and their extensions [30, 35]. In the context of vulnerability repair, ROUGE scores evaluate the patch based on the number of occurrences of n-grams from the known repaired code (reference sequence) in the patch (generated sequence). By contrast, BLEU shows n-gram precision of


We employ different metrics for evaluating the repair suggestions.

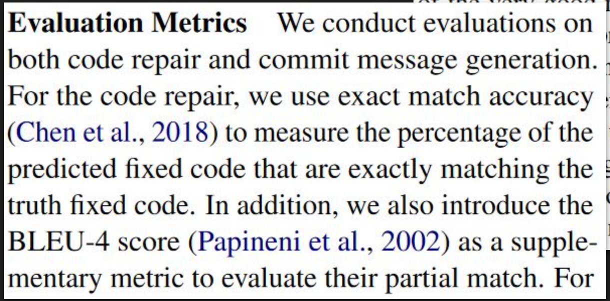
Perplexity. Perplexity [4] measures how well a model predicts samples. Low (e.g., single digit) perplexity values indicate the model is good at predicting a given sequence.

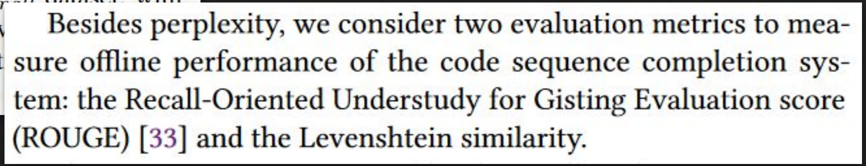
BLEU. The next metric we use to assess the generated output of the model is BLEU [37]. BLEU is a well-known and popular metric for automatically evaluating the quality of machine-translated sentences. It has been shown to correlate well with human judgments [7, 17]. BLEU calculates how well a given sequence is matched with an expected sequence in terms of the actual tokens and their ordering using an n-gram model. The output of the BLEU metric is a number between 1–100. For natural language translations, BLEU scores of 25–40 are considered high scores [47].

Syntactic Validation. For validating the suggestions for syntactical correctness, we generate a lexer and parser from our Delta grammar through ANTLR4. We pass each inferred suggestion through the Delta lexer/parser. This way, we assess whether the model generates suggestions that conform to the grammar of the expected resolution changes. The output is binary, i.e., either the suggestion is valid or invalid.


B. Injection of Code Mutants (MG)

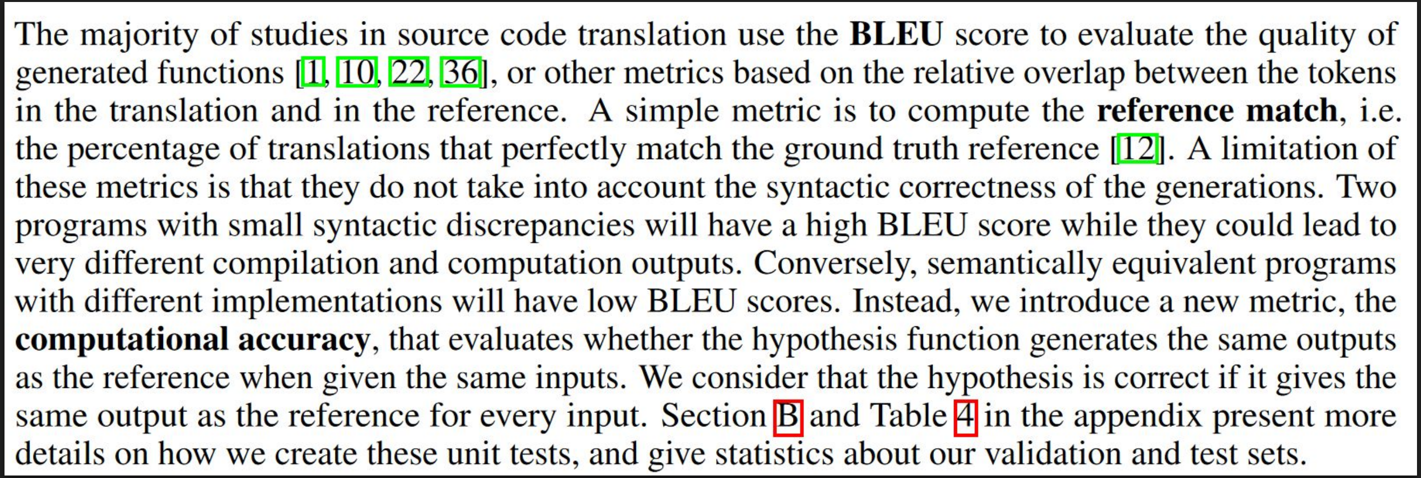
Looking at Fig. 3, we can observe that using T5 to generate mutants allows to obtain much more accurate results than the baseline, with the Accuracy@1 improving by 11%, with 1,240 additional perfect predictions (+62% as compared to the baseline). The average BLEU score improves by ∼0.01 on top
results already obtained by the baseline (*i.e.*,
rovements in BLEU score can still indicate
the quality of the generated solutions [69].
rence time (Table VI), we observed similar
to the BF task on the BF_{small} dataset: with
ge inference time is 0.31s, w
o not report perfect predict
not reported in the original


Evaluation Metrics We conduct evaluations on both code repair and commit message generation. For the code repair, we use exact match accuracy (Chen et al., 2018) to measure the percentage of the predicted fixed code that are exactly matching the truth fixed code. In addition, we also introduce the BLEU-4 score (Papineni et al., 2002) as a supplementary metric to evaluate their partial match. For


Besides perplexity, we consider two evaluation metrics to measure offline performance of the code sequence completion system: the Recall-Oriented Understudy for Gisting Evaluation score (ROUGE) [33] and the Levenshtein similarity.

Program Repair and Improvement Without Tests?

- One of the first papers to use such models but also consider running the resulting code against tests was Facebook's TransCoder (9/2020)



The majority of studies in source code translation use the **BLEU** score to evaluate the quality of generated functions [1, 10, 22, 36], or other metrics based on the relative overlap between the tokens in the translation and in the reference. A simple metric is to compute the **reference match**, i.e. the percentage of translations that perfectly match the ground truth reference [12]. A limitation of these metrics is that they do not take into account the syntactic correctness of the generations. Two programs with small syntactic discrepancies will have a high BLEU score while they could lead to very different compilation and computation outputs. Conversely, semantically equivalent programs with different implementations will have low BLEU scores. Instead, we introduce a new metric, the **computational accuracy**, that evaluates whether the hypothesis function generates the same outputs as the reference when given the same inputs. We consider that the hypothesis is correct if it gives the same output as the reference for every input. Section B and Table 4 in the appendix present more details on how we create these unit tests, and give statistics about our validation and test sets.

- From the language model perspective, tests were novel and uncommon

GPT Evaluation With Tests?

- While TransCoder is a different problem (translation, not repair or improvement), the “computational accuracy” of 25-75%

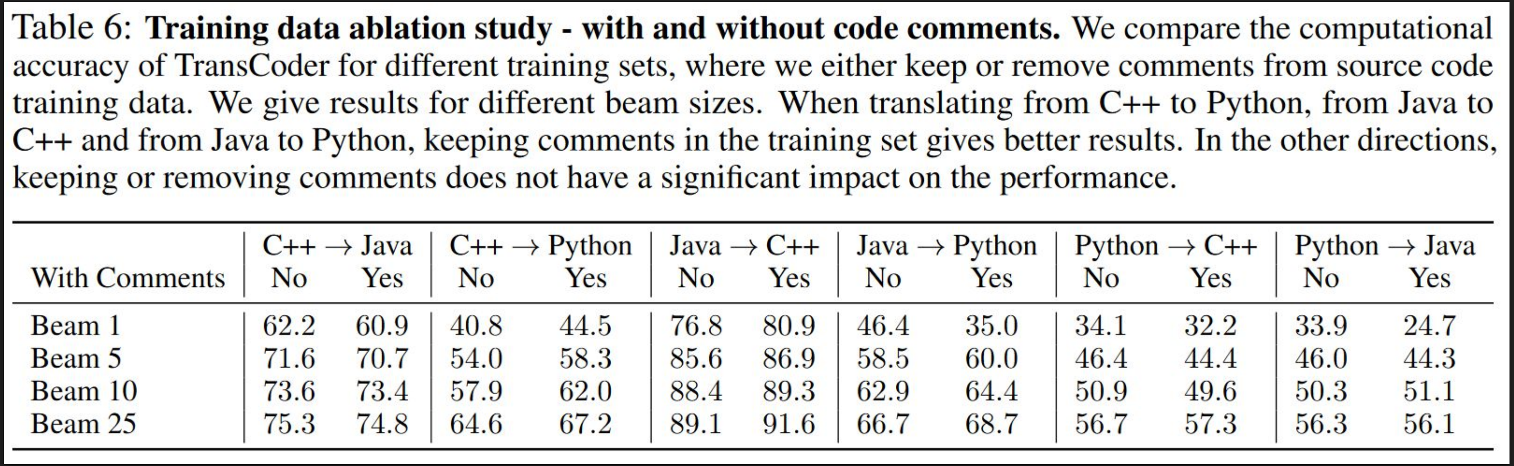


Table 6: **Training data ablation study - with and without code comments.** We compare the computational accuracy of TransCoder for different training sets, where we either keep or remove comments from source code training data. We give results for different beam sizes. When translating from C++ to Python, from Java to C++ and from Java to Python, keeping comments in the training set gives better results. In the other directions, keeping or removing comments does not have a significant impact on the performance.

	C++ → Java		C++ → Python		Java → C++		Java → Python		Python → C++		Python → Java	
With Comments	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
Beam 1	62.2	60.9	40.8	44.5	76.8	80.9	46.4	35.0	34.1	32.2	33.9	24.7
Beam 5	71.6	70.7	54.0	58.3	85.6	86.9	58.5	60.0	46.4	44.4	46.0	44.3
Beam 10	73.6	73.4	57.9	62.0	88.4	89.3	62.9	64.4	50.9	49.6	50.3	51.1
Beam 25	75.3	74.8	64.6	67.2	89.1	91.6	66.7	68.7	56.7	57.3	56.3	56.1

- … is more like what we see from non-GPT program repair

The Lens of Construct Validity

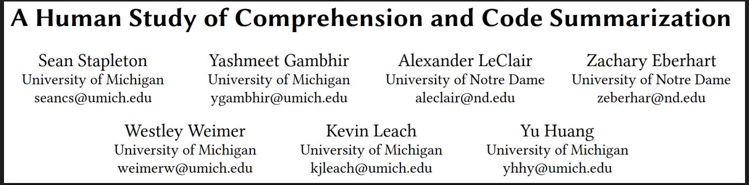
- Construct validity is the appropriateness of inferences made on the basis of observations or measurements (often test scores), specifically whether a test can reasonably be considered to reflect the intended construct. It subsumes content and criterion validity.
    - In this context, informally: are you measuring what you say you're measuring?
- Example: You conduct a human study in which you show participants snippets of code and ask them comprehension questions. You use their times and accuracies to make inferences about code readability. However, a threat to the construct validity of those results relates to whether you are measuring *readability* or *complexity*.

Two Countries Divided By A Common Language

- Approach X is better than approach Y at the program repair task
  - Better than
    - “Produces token sequences yielding higher ROUGE (etc.) scores w.r.t. a reference”
    - “Produces more patches that pass all test cases”
  - Program repair task
    - “Given a program prefix and perfect fault localization and a large trained model, produce a patch using previous code”
    - “Given a program and test cases, produce a patch that possibly uses new code”
- Informally, one anxiety making the rounds in our community is that program committees and grant panels may be too inundated to make the distinctions
  - And thus mistakenly conclude that a claim about “Better_definition1” is really a claim about “Better_definition2”, etc.

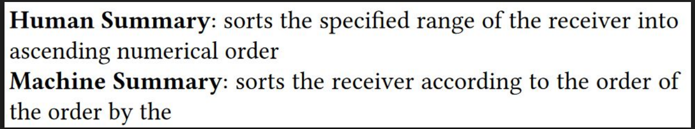
Recommendation: More Human Studies

- We evaluated a state-of-the-art encoder-decoder model via a human study of 45 professionals and students
- Metrics like BLEU did not necessarily match human intuition
  - In the example on the right, the summary has a moderately high score
- Participants performed significantly better (p = 0.029) using human-written summaries versus machine-generated summaries
- Participants' performance showed no correlation with the BLEU and ROUGE scores often used to assess the quality of machine-generated summaries

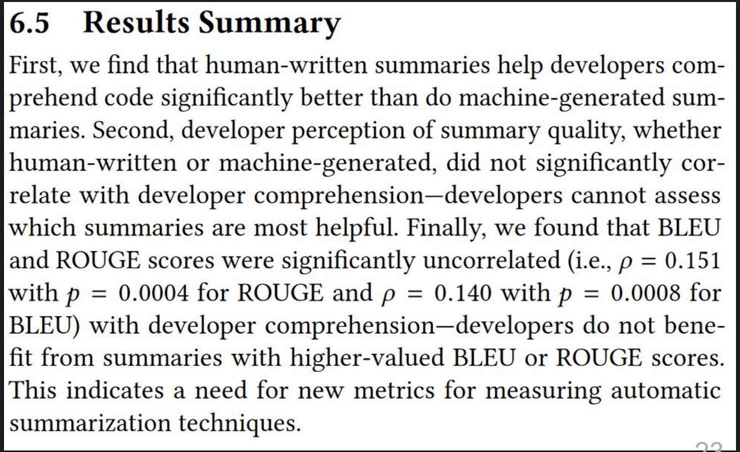


A Human Study of Comprehension and Code Summarization

Sean Stapleton, University of Michigan, seancs@umich.edu
Yashmeet Gambhir, University of Michigan, ygambhir@umich.edu
Alexander LeClair, University of Notre Dame, aleclair@nd.edu
Zachary Eberhart, University of Notre Dame, zeberhar@nd.edu
Westley Weimer, University of Michigan, weimerw@umich.edu
Kevin Leach, University of Michigan, kjleach@umich.edu
Yu Huang, University of Michigan, yhhy@umich.edu



Human Summary: sorts the specified range of the receiver into ascending numerical order
Machine Summary: sorts the receiver according to the order of the order by the

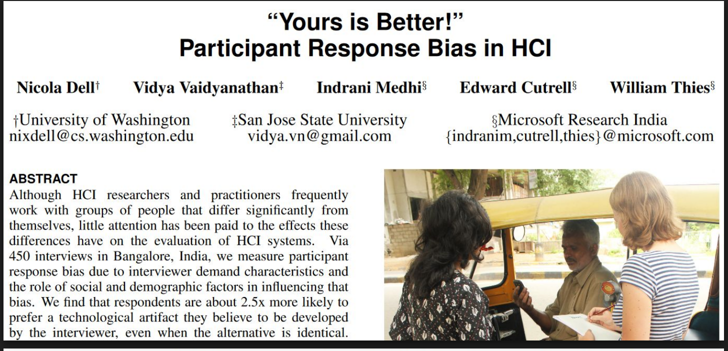


6.5 Results Summary

First, we find that human-written summaries help developers comprehend code significantly better than do machine-generated summaries. Second, developer perception of summary quality, whether human-written or machine-generated, did not significantly correlate with developer comprehension—developers cannot assess which summaries are most helpful. Finally, we found that BLEU and ROUGE scores were significantly uncorrelated (i.e., $\rho = 0.151$ with $p = 0.0004$ for ROUGE and $\rho = 0.140$ with $p = 0.0008$ for BLEU) with developer comprehension—developers do not benefit from summaries with higher-valued BLEU or ROUGE scores. This indicates a need for new metrics for measuring automatic summarization techniques.

Recommendation: Human Studies, Deception, Context

- One challenge in comparative human studies is that non-anonymized presentations may result in bias
- A human study may employ deception (e.g., describing a patch as written by a human instead of a machine, or vice-versa), with a debriefing
- Alternatively, real-world contexts, such as deployments on GitHub, provide end-to-end assessments



"Yours is Better!"
Participant Response Bias in HCI

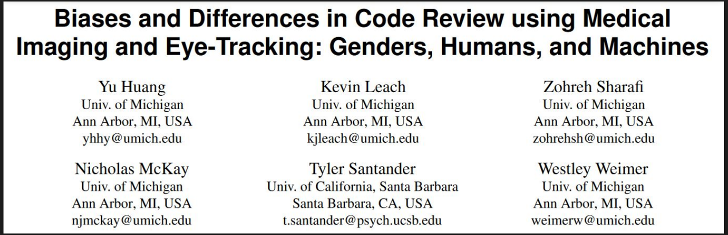
Nicola Dell[†] **Vidya Vaidyanathan**[‡] **Indrani Medhi**[§] **Edward Cutrell**[§] **William Thies**[§]

[†]University of Washington
nixdell@cs.washington.edu

[‡]San Jose State University
vidya.vn@gmail.com

[§]Microsoft Research India
{indranim,cutrell,thies}@microsoft.com

ABSTRACT
Although HCI researchers and practitioners frequently work with groups of people that differ significantly from themselves, little attention has been paid to the effects these differences have on the evaluation of HCI systems. Via 450 interviews in Bangalore, India, we measure participant response bias due to interviewer demand characteristics and the role of social and demographic factors in influencing that bias. We find that respondents are about 2.5x more likely to prefer a technological artifact they believe to be developed by the interviewer, even when the alternative is identical.



Biases and Differences in Code Review using Medical Imaging and Eye-Tracking: Genders, Humans, and Machines

Yu Huang
Univ. of Michigan
Ann Arbor, MI, USA
yhhy@umich.edu

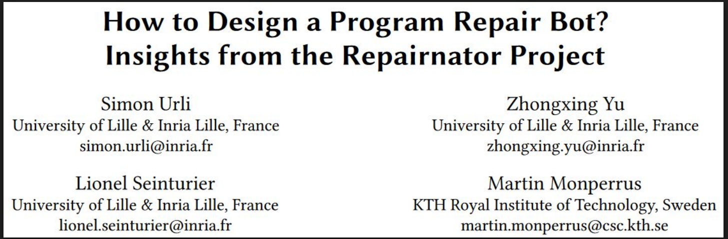
Kevin Leach
Univ. of Michigan
Ann Arbor, MI, USA
kjleach@umich.edu

Zohreh Sharafi
Univ. of Michigan
Ann Arbor, MI, USA
zohrehsh@umich.edu

Nicholas McKay
Univ. of Michigan
Ann Arbor, MI, USA
njmckay@umich.edu

Tyler Santander
Univ. of California, Santa Barbara
Santa Barbara, CA, USA
t.santander@psych.ucsb.edu

Westley Weimer
Univ. of Michigan
Ann Arbor, MI, USA
weimerw@umich.edu



How to Design a Program Repair Bot?
Insights from the Repairnator Project

Simon Urli
University of Lille & Inria Lille, France
simon.urli@inria.fr

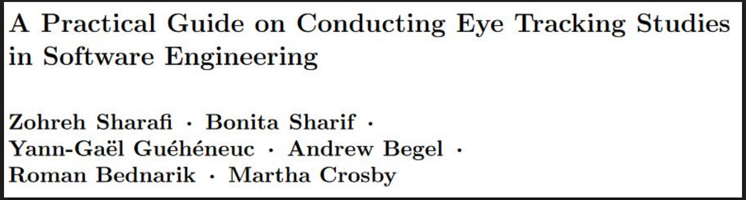
Zhongxing Yu
University of Lille & Inria Lille, France
zhongxing.yu@inria.fr

Lionel Seinturier
University of Lille & Inria Lille, France
lionel.seinturier@inria.fr

Martin Monperrus
KTH Royal Institute of Technology, Sweden
martin.monperrus@csc.kth.se

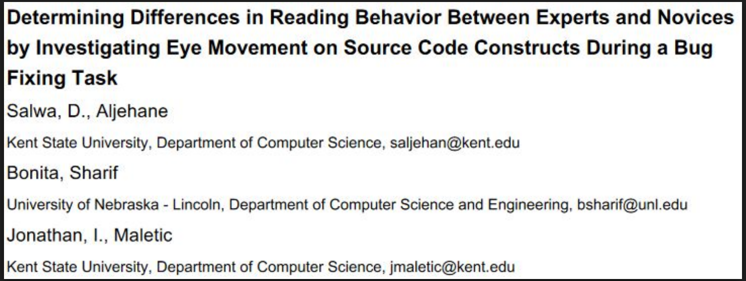
Recommendation: Eye Tracking

- Eye tracking is becoming an increasingly common addition to human studies
  - The equipment is inexpensive
  - It can often detect where attention is paid at the level of individual words or syntax
  - It provides a validated way of assessing cognitive load (via pupil dilation, etc.)
- As deep learning models produce code or text, and as NLP metrics largely ignore semantics, measuring where humans pay attention is quite relevant



A Practical Guide on Conducting Eye Tracking Studies in Software Engineering

Zohreh Sharafi · Bonita Sharif · Yann-Gaël Guéhéneuc · Andrew Begel · Roman Bednarik · Martha Crosby



Determining Differences in Reading Behavior Between Experts and Novices by Investigating Eye Movement on Source Code Constructs During a Bug Fixing Task

Salwa, D., Aljehane

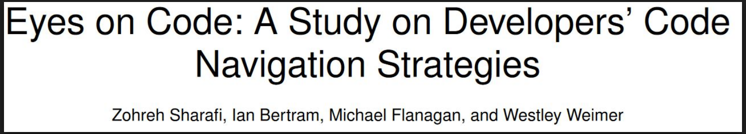
Kent State University, Department of Computer Science, saljehan@kent.edu

Bonita, Sharif

University of Nebraska - Lincoln, Department of Computer Science and Engineering, bsharif@unl.edu

Jonathan, I., Maletic

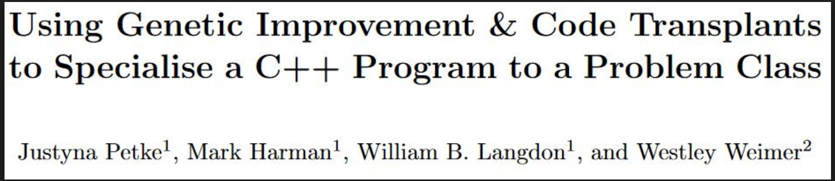
Kent State University, Department of Computer Science, jmaletic@kent.edu



Eyes on Code: A Study on Developers' Code Navigation Strategies

Zohreh Sharafi, Ian Bertram, Michael Flanagan, and Westley Weimer

Recommendation: Algorithms


Using Genetic Improvement & Code Transplants to Specialise a C++ Program to a Problem Class

Justyna Petke[1], Mark Harman[1], William B. Langdon[1], and Westley Weimer[2]

- In code summarization work, we used an "encoder + decoder + additional encoder for the AST" model to incorporate program structure
- Such AST-inclusive approaches may form a natural bridge to the grammar-based GP work of Langdon and others
- We need algorithms to take the output of deep learning models (e.g., Copilot) and improve it
- We might focus on the synthesis and discovery of novel code, leaving simple bugs that can be fixed with existing ingredients to AI
  - Just as we may not leave null pointer errors to program repair approaches
- Target fault localization for transformer approaches or, dually, target domains for which perfect fault localization is unreasonable

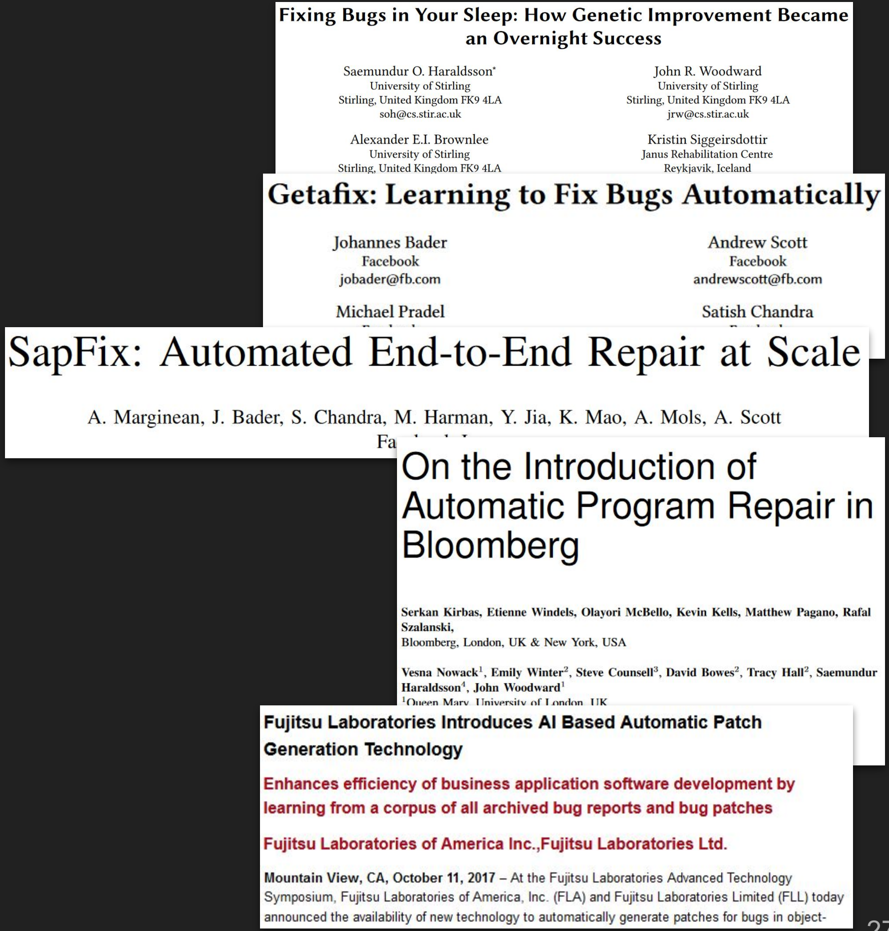
Program Repair Deployments

Janus Manager (2017): smaller, fixes Python Exceptions

Facebook SapFix, Getafix (2018-19): 60MLOC+, mostly Null Pointer Exceptions

Bloomberg (2021): uninitialized variables, other templates, 48% dev accept rate

Fujitsu (2016-2017): method invocation bugs, ~50% acceptance rate, reduces dev time by ~29%



Fixing Bugs in Your Sleep: How Genetic Improvement Became an Overnight Success

Saemundur O. Haraldsson* University of Stirling Stirling, United Kingdom FK9 4LA soh@cs.stir.ac.uk

John R. Woodward University of Stirling Stirling, United Kingdom FK9 4LA jrw@cs.stir.ac.uk

Alexander E.I. Brownlee University of Stirling Stirling, United Kingdom FK9 4LA

Kristin Siggeirsdottir Janus Rehabilitation Centre Reykjavik, Iceland

Getafix: Learning to Fix Bugs Automatically

Johannes Bader Facebook jobader@fb.com

Andrew Scott Facebook andrewscott@fb.com

Michael Pradel

Satish Chandra

SapFix: Automated End-to-End Repair at Scale

A. Marginean, J. Bader, S. Chandra, M. Harman, Y. Jia, K. Mao, A. Mols, A. Scott

Fa

On the Introduction of Automatic Program Repair in Bloomberg

Serkan Kirbas, Etienne Windels, Olayori McBello, Kevin Kells, Matthew Pagano, Rafal Szalanski,
Bloomberg, London, UK & New York, USA

Vesna Nowack[1], Emily Winter[2], Steve Counsell[3], David Bowes[2], Tracy Hall[2], Saemundur Haraldsson[4], John Woodward[1]
[1]Queen Mary, University of London, UK

Fujitsu Laboratories Introduces AI Based Automatic Patch Generation Technology

Enhances efficiency of business application software development by learning from a corpus of all archived bug reports and bug patches

Fujitsu Laboratories of America Inc.,Fujitsu Laboratories Ltd.

Mountain View, CA, October 11, 2017 – At the Fujitsu Laboratories Advanced Technology Symposium, Fujitsu Laboratories of America, Inc. (FLA) and Fujitsu Laboratories Limited (FLL) today announced the availability of new technology to automatically generate patches for bugs in object-

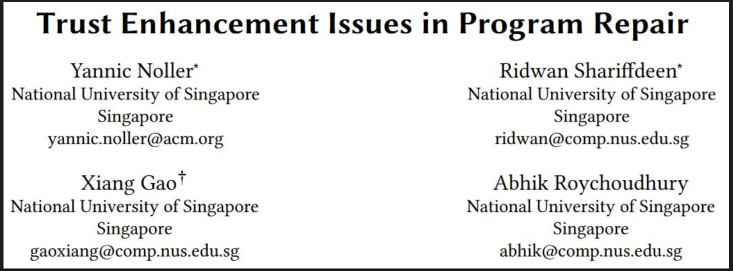
Deployment Commonalities

- Most focus on a single type of defect (e.g., Null Pointer Exceptions, OO Method Invocation errors, etc.) via fix patterns
  - For example, while Getafix handles multiple types of bugs, 804/1264 were Null Pointer Exceptions
- “Bloomberg views the readability of a fix and future-proofing of fixes as a fundamental and crucial part of the overall repair process”
- Acceptance rates are uniform: ~50% at Bloomberg, Facebook, and Fujitsu
- Potential implication: near-future deployments may not require >50% success rate and may favor readability and simplicity

Trust and Acceptability

- Surveying 100 developers, Noller et al. found that manual review and test cases were critical to acceptancing of APR

- The emphasis on manual review motivates the inclusion of human studies (including advanced approaches like eye tracking or deception) in evaluations
- The emphasis on test cases motivates the nuanced use of extrinsic metrics in evaluations



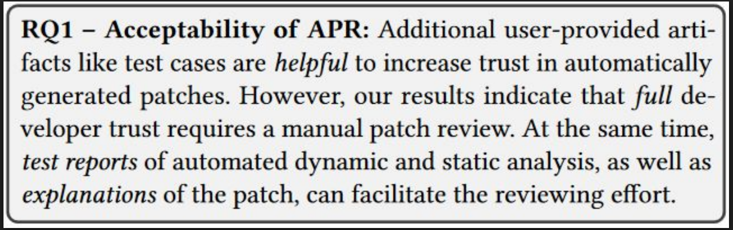
Trust Enhancement Issues in Program Repair

Yannic Noller[*]
National University of Singapore
Singapore
yannic.noller@acm.org

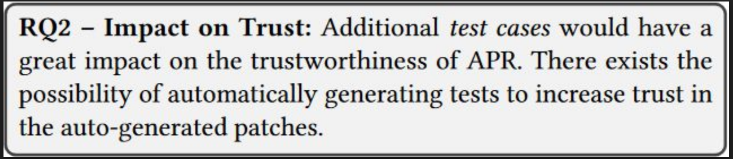
Ridwan Shariffdeen[*]
National University of Singapore
Singapore
ridwan@comp.nus.edu.sg

Xiang Gao[†]
National University of Singapore
Singapore
gaoxiang@comp.nus.edu.sg

Abhik Roychoudhury
National University of Singapore
Singapore
abhik@comp.nus.edu.sg



RQ1 – Acceptability of APR: Additional user-provided artifacts like test cases are *helpful* to increase trust in automatically generated patches. However, our results indicate that *full* developer trust requires a manual patch review. At the same time, *test reports* of automated dynamic and static analysis, as well as *explanations* of the patch, can facilitate the reviewing effort.



RQ2 – Impact on Trust: Additional *test cases* would have a great impact on the trustworthiness of APR. There exists the possibility of automatically generating tests to increase trust in the auto-generated patches.

Summary

- The application of neural network deep learning language models to program improvement, completion and repair tasks has surged
  - Codex, Copilot, GPT, Transcoder, etc., are popular examples
- Concerns and challenges abound
  - Training costs may be exclusionary, novel synthesis is uncertain, perfect fault localization is often assumed, and intrinsic metrics omit semantics (such as running the program)
  - Informally, there is a fear that PCs and PMs will misinterpret results
- At the same time, opportunities exist
  - How we conduct peer review, clarity of communication, human studies (e.g., eye tracking and deception), and algorithmic advances (e.g., grammars, novelty, FL)
  - Real-world deployments focus on simplicity, humans reading patches, tests, and trust
    - Perhaps fear has misdirected our recent attention away from end-user needs