A Fast Low-Level Error Detection Technique

Zhengyang He, Hui Xu, Guanpeng Li

Reza Adinepour

Amirkabir University of Technology
(Tehran Polytechnic)

Computer Engineering Department
January 13, 2025

Agenda

Problem & Solutions Overview

- **Problem:** Transient hardware faults (soft errors) due to shrinking transistor sizes and operating voltages.
- **Impact:** Soft errors can cause Silent Data Corruptions (SDCs), compromising system dependability.
- **Solutions:**
  1. Traditional: Hardware-based methods such as:
     - voltage guard bands
     - redundancy

     have high overhead in performance and energy consumption.
  2. Software-Based: Error Detection by Duplicating Instructions (EDDI)

     has been proposed as a flexible, resource-efficient alternative.

EDDI Methods

- **EDDI:** Duplicates instructions at compile time and checks for mismatches at runtime.

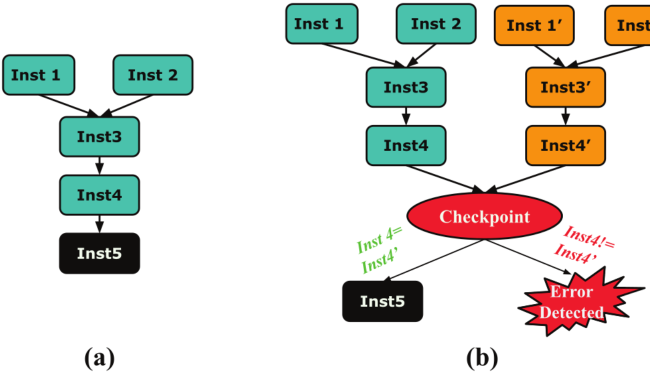


Figure: High-level idea of EDDI

EDDI Methods (Cont.)

- **Existing EDDI Methods:**
  1. Mostly at IR level

  reduced fault coverage when tested at the assembly level.
- **Problem with IR-Level EDDI:**
  - Fault coverage gaps at IR level.
  - Reduced effectiveness when evaluated at assembly level.
  - Underestimated error detection at lower levels.
  - Need for assembly-level implementation for better fault protection.

IR Code Example Using EDDI

```
// High-level C code
int add(int a, int b) {
    return a + b;
}
```

```
define i32 @add(i32 %a, i32 %b) {
entry:
  %a.addr = alloca i32, align 4
  %b.addr = alloca i32, align 4
  store i32 %a, i32* %a.addr, align 4
  store i32 %b, i32* %b.addr, align 4
;Duplicate instruction
  %0 = load i32, i32* %a.addr, align 4
  %1 = load i32, i32* %a.addr, align 4
;Duplicate instruction
  %2 = load i32, i32* %b.addr, align 4
```

Figure: (a)

```
  %3 = load i32, i32* %b.addr, align 4
;Duplicate instruction
  %add = add nsw i32 %0, %1
  %add2 = add nsw i32 %2, %3
;Check the results
  %cmp = icmp eq i8** %add, %add2
  br i1 %cmp, label %4, label %checkBb

checkBb:
  call void @check_flag()
  br label %4

<label>:4
  ret i32 %add
}
```

Figure: (b)

Main Contribution

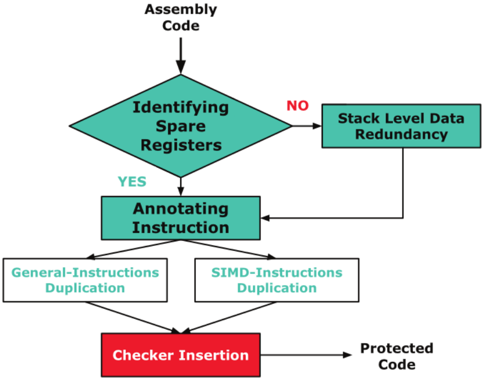
1. **Proposed Solution:**
   - FERRUM: Optimized assembly-level EDDI.
   - Enhancements: Utilizes SIMD and compiler optimizations.
   - Improves: Fault coverage and performance.
2. **Key Findings & Results:**
   - 28% gap in fault coverage (IR-level vs. assembly-level).
   - 100% fault coverage with FERRUM at assembly level.
   - 52% reduction in runtime overhead with FERRUM, no loss in fault coverage.

Background

1. **Focus on single bit-flip transient faults in:**
   - Processor computing components
   - Pipeline stages
   - Arithmetic components
   - Load/store units

   Do not consider faults in the memory or caches, as we assume they have already been protected by ECC (Error Correcting Code).
2. **Fault Simulation:** Assembly-level fault injection; beam testing infeasible.
3. **EDDI:** Instruction duplication, runtime comparison.
4. **Platform:** x86 ISA (other platforms for future work).

High-Level Design



- Scan registers (general-purpose, SIMD); identify spare registers.
- Annotate instructions for SIMD compatibility.
- Duplicate instructions; use SIMD or general-purpose registers; fallback to stack if needed.

Components

1. **Static Code Analysis**
   - Identify spare registers (general-purpose: 2, SIMD: 4 XMM).
   - Annotate instructions (SIMD-enabled or general).
2. **Duplication for General Instructions**
   - Duplicate instructions; use spare registers or deferred detection for comparisons (e.g., `rflag`).
3. **Duplication for SIMD-Enabled Instructions**
   - Use SIMD registers (e.g., XMM, YMM) for bulk comparison.
   - Leverage architecture-specific features (e.g., ZMM on Intel CPUs).
4. **Stack-Level Data Redundancy**
   - Buffer unused registers onto stack when spare registers are insufficient.
   - Restore registers after duplication and checking.

Example1

```
.LBB0_3:
...
movslq  %ecx, %r10
movslq  %ecx, %rcx #original instruction
xorq    %rcx, %r10
jne exit_function
...
```

Figure: Protection of GENERAL-INSTRUCTIONS (`movslq`)

Example2

```
BB1:
movq        -24(%rbp), %xmm0
movq        -24(%rbp), %rax #original Ins
movq        %rax, %xmm1
pinsrq      $1, 8(%rax), %xmm0
movq        8(%rax), %rdi   #original Ins
pinsrq      $1, %rdi, %xmm1
...
movq        -24(%rbp), %xmm2
movq        -24(%rbp), %rax #original Ins
movq        %rax, %xmm3
pinsrq      $1, 16(%rax), %xmm2
movq        16(%rax), %rdi  #original Ins
pinsrq      $1, %rdi, %xmm3
vinserti128 $1, %xmm2, %ymm0, %ymm0
vinserti128 $1, %xmm3, %ymm1, %ymm1
vpxor       %ymm1, %ymm0, %ymm0
vptest      %ymm0, %ymm0
jne exit_function
...
```

Figure: FERRUM using SIMD capability

Experimental Setup

Table: Details of Benchmarks

Benchmark	Suite	Domain
Backprop	Rodinia	Machine Learning
BFS	Rodinia	Graph Algorithm
Pathfinder	Rodinia	Dynamic Programming
LUD	Rodinia	Linear Algebra
Needle	Rodinia	Dynamic Programming
kNN	Rodinia	Machine Learning
kmeans	Rodinia	Data Mining
Particlefilter	Rodinia	Noise estimator

- Platform: Ubuntu 20.04, Intel Xeon (x86-64), 64GB RAM.

Fault Injection Methodology

1. **Single bit-flip faults** injected at assembly level.
2. **1000 random faults** injected per benchmark.
1. **Metrics:**
   - SDC Coverage: Measures reduction in Silent Data Corruptions.

$$Coverage = \frac{SDC_{raw} - SDC_{prot}}{SDC_{raw}}$$

   - Runtime Overhead: Measures performance impact.

$$Overhead = \frac{Runtime_{prot} - Runtime_{raw}}{Runtime_{raw}}$$

   - FERRUM Execution Time: Compile-time overhead.

SDC Coverage

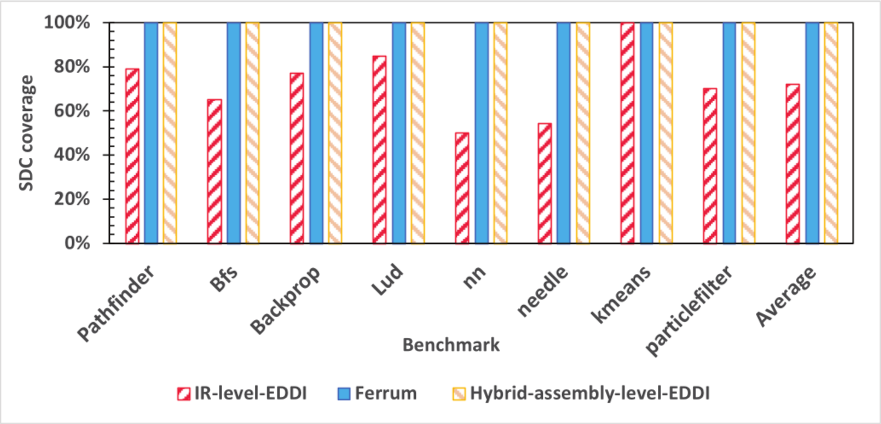


Figure: SDC coverage measured

Runtime Performance Overhead

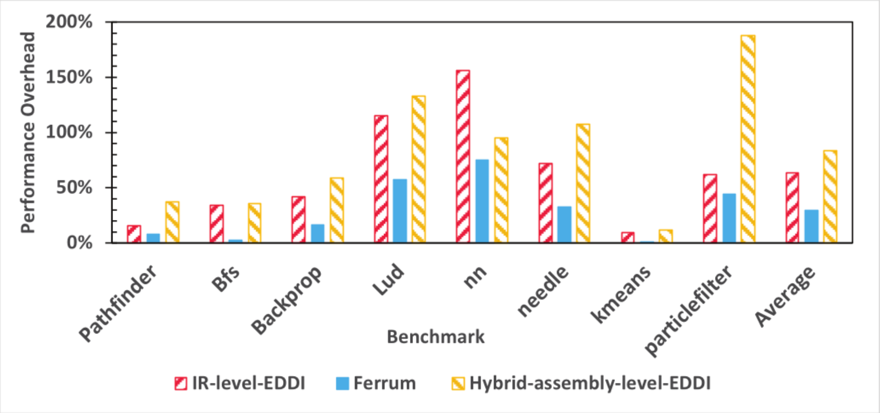


Figure: Performance overhead measured

Execution Time

1. Average: 0.117 seconds.
2. Max: 0.196 seconds.
3. Min: 0.089 seconds (BFS).

References

Zhengyang He, Hui Xu, Guanpeng Li (2024)
A Fast Low-Level Error Detection Technique
2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), University of Iowa, Iowa City, IA, USA; Fudan University, Shanghai, China.

The End

Questions? Comments?

adinepour@aut.ac.ir

PagPassGPT

Pattern Guided Password Guessing via Generative Pretrained Transformer

Reza Adinepour

Amirkabir University of Technology
(Tehran Polytechnic)

Computer Engineering Department

January 13, 2025

Agenda

Problem & Solutions Overview

1. **Problem:** Deep learning-based password guessing models face challenges in:
    - Generating high-quality passwords.
    - Reducing the rate of duplicate passwords.
2. **Impact:** Reduced efficiency in password guessing models due to:
    - Lower hit rates.
    - High redundancy in generated passwords, limiting practical effectiveness.

Solutions

1. PagPassGPT:
   - Built on a Generative Pretrained Transformer (GPT).
   - Incorporates pattern structure information as background knowledge to improve guessing accuracy.
2. D&C-GEN (Divide-and-Conquer Generation):
   - Divides password guessing tasks into non-overlapping subtasks.
   - Subtasks inherit parent task knowledge for efficient prediction.
   - Effectively reduces duplicate passwords.
3. Results:
   - 12% higher hit rate compared to state-of-the-art models.
   - 25% fewer duplicate passwords.

Main Contribution

1. **PagPassGPT:**
   - Combines password patterns with deep learning.
   - Improves guessing accuracy.
2. **D&C-GEN:**
   - Uses divide-and-conquer for task splitting.
   - Reduces duplicate passwords.
3. **Performance:**
   - Validated on public datasets.
   - Outperforms state-of-the-art models in hit rate and duplicates.

Related works

1. **Password Guessing Types**
   - Trawling Attack:
     Problem: Misses rate patterns; requires accurate modeling.
   - Targeted Attack:
     Problem: Depends on personally identifiable information (PII); less effective with unpredictable users.
2. **Password Guessing Models**
   - Rule-based Models:
     Problem: Background knowledge dependency; limited rules.
   - Probability-based Models:
     Problem: Fixed vocabulary; poor segmentation accuracy.
3. **Deep Learning-based Models:**
   Problem: Accuracy loss; high computation.

PagPass Methods

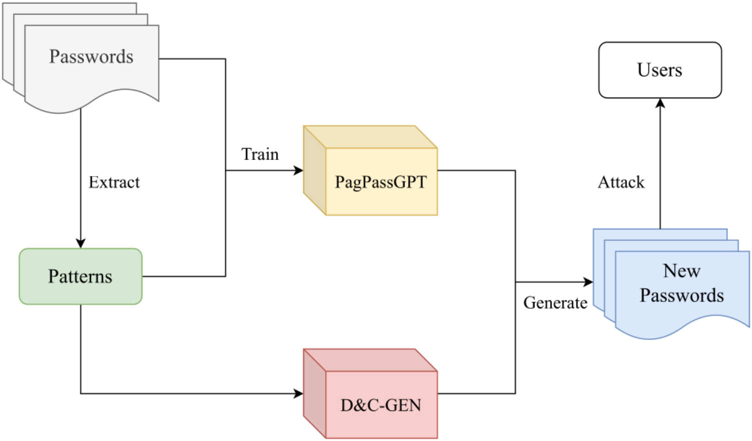


Figure: High-level idea of EDDI

Training Process

1. **Input:** Passwords from a training dataset.
2. **Training:**
   - Extract password patterns (e.g., "L4N3S1") using PCFG rules.
   - Combine patterns and passwords into a structured sequence: `<BOS> || Pattern || <SEP> || Password || <EOS>`
   - Tokenize sequences and embed using GPT-2 architecture.
   - Optimize with cross-entropy loss for improved prediction accuracy.

Training Process (Cont.)

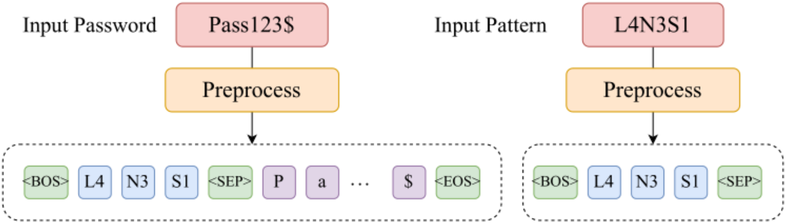


Figure: The preprocessing operation of tokenizer of PagPassGPT

Training Process (Cont.)

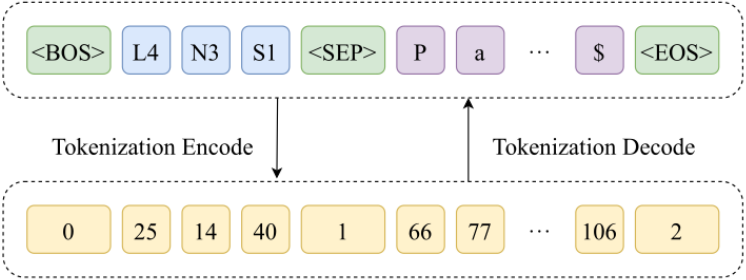


Figure: The tokenization process of the tokenizer of PagPassGPT

Training Process (Cont.)

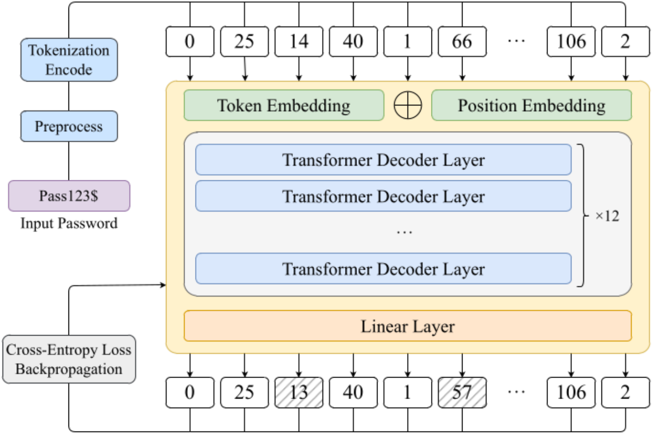


Figure: Training process architecture

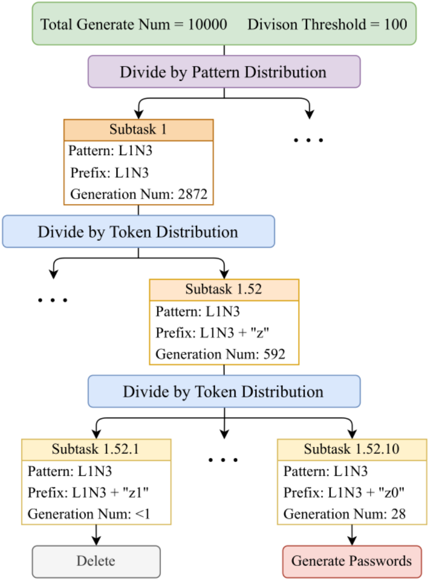
Training Process (Cont.)

1. **Generation:**
   - Predict tokens sequentially using an auto-regressive mechanism based on:
     - Historical tokens.
     - Password patterns.
   - Achieves 27.5% higher hit rate compared to PassGPT.

D&C-GEN

1. **Objective:** Reduce duplicate passwords using divide-and-conquer.
2. **Workflow:**
   - Split tasks into non-overlapping subtasks by patterns and prefixes.
   - Apply a threshold T to stop division and execute generation.
   - Generate passwords efficiently under task constraints.
3. **Performance:**
   - Reduces duplicate rate to 9.28% for 10^9 guesses.
   - Supports parallel execution and optimized GPU utility.

D&C-GEN (Cont.)


Total Generate Num = 10000 Divison Threshold = 100
Divide by Pattern Distribution
Subtask 1
Pattern: L1N3
Prefix: L1N3
Generation Num: 2872
Divide by Token Distribution
Subtask 1.52
Pattern: L1N3
Prefix: L1N3 + "z"
Generation Num: 592
Divide by Token Distribution
Subtask 1.52.1
Pattern: L1N3
Prefix: L1N3 + "z1"
Generation Num: <1
Subtask 1.52.10
Pattern: L1N3
Prefix: L1N3 + "z0"
Generation Num: 28
Delete
Generate Passwords

Evaluation

1. **Datasets**
   - Ethical Considerations:
     - Public data, minimal usage, and strictly for research purposes.
   - Applied Datasets:
     - RockYou, LinkedIn, phpBB, MySpace, Yahoo!
     - Total entries: 75,349,874.
   - Data Cleaning:
     - Password length: 4–12 characters.
     - Removed duplicates and non-ASCII characters.
   - Data Utilization:
     - RockYou & LinkedIn: Split into training (70%), validation (10%), and testing (20%).
     - Cross-site evaluation: Used all remaining datasets.

Evaluation (Cont.)

1. **Models**
   - PagPassGPT:
     - Trained with batch size 512 for 30 epochs using AdamW optimizer.
     - Max tokens: 32, Embedding size: 256.
     - Hidden layers: 12, Attention heads: 8.
     - Training duration: 25+ hours on 4 RTX 3080 GPUs.
2. **Drawling Attack Test**
   - Setup:
     - Compared PagPassGPT and PagPassGPT-D&C (with threshold $T = 4000$) against models like PassGAN, VAEPass, PassFlow, and PassGPT.

Evaluation (Cont.)

1. **Metrics**
   - Hit Rate:
     - Ratio of correctly guessed passwords to total test set passwords.
     - PagPassGPT-D&C achieved a 53.63% hit rate for 10^9 guesses, 12% higher than PassGPT.
   - Repeat Rate:
     - Reflects duplicate passwords among generated ones.
     - PagPassGPT-D&C achieved a 9.28% repeat rate, significantly lower than PassGPT's 34.5%.

Hit Rate

Table: Hit rates of different models in trawling attack test.

Guess Num	10^6	10^7	10^8	10^9
PassGAN	0.80%	3.11%	8.24%	16.32%
VAEPass	0.49%	2.24%	6.24%	12.23%
PassFlow	0.26%	1.62%	7.03%	14.10%
PassGPT	0.73%	5.60%	21.43%	41.93%
PagPassGPT	1.00%	7.68%	27.23%	48.75%
PagPassGPT-D&C	**1.05%**	**8.48%**	**31.38%**	**53.63%**

Repeat Rates

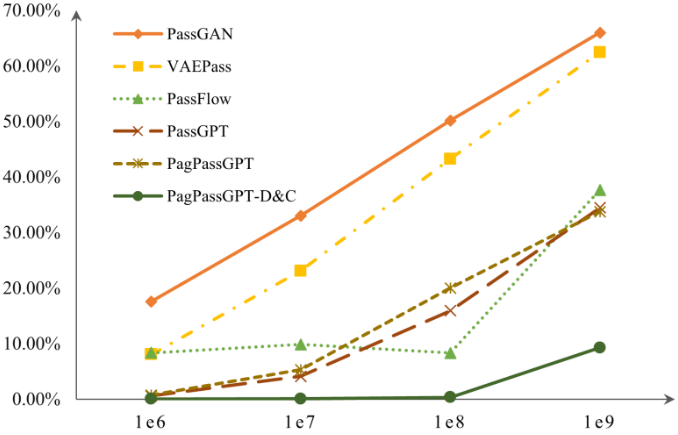


Figure: Repeat rates of passwords generated by different models

References

Xingyu Su, Xiaojie Zhu, Yang Li, Yong Li, Chi Chen, Paulo Esteves-Veríssimo (2024)
PagPassGPT: Pattern Guided Password Guessing via Generative Pretrained Transformer
arXiv:2404.04886v2 [cs.CR], School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China; Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China; King Abdullah University of Science and Technology, Thuwal, Kingdom of Saudi Arabia.
Emails: {suxingyu, liyang8119, liyong, chenchi}@iie.ac.cn, {xiaojie.zhu, paulo.verissimo}@kaust.edu.sa.

The End

Questions? Comments?

adinepour@aut.ac.ir