National College of Ireland

Configuration Manual

MSc Research Project
Cyber Security

Shreyas Unhale
Student ID: 23267747

School of Computing
National College of Ireland

Supervisor: Jawad Salahuddin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shreyas Unhale
Student ID:	23267747
Programme:	MSc in Cybersecurity **Year:** 2024-2025
Module:	MSc Research Project
Lecturer:	Jawad Salahuddin
Submission Due Date:	11/08/2025
Project Title:	Evaluating the Effectiveness of NIST 800-63B Password Policies Against Open-Source Cracking Tools
Word Count:	513 **Page Count:** 10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Shreyas Unhale

Date: 11/08/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Shreyas Unhale
Student ID: 23267747

1 System Properties

1.1 Host and Virtualization

- **Host operating system**

  - **Windows edition:** Windows 11 Pro/Enterprise (build and edition)

Figure 1 : Device Specifications

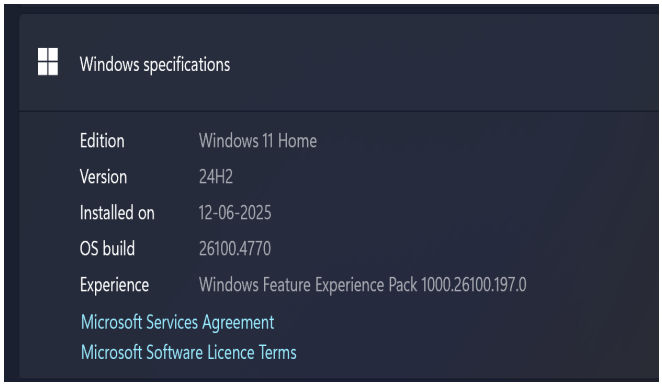

Figure 2: Windows Specifications

- **Virtualization**: WSL2 enabled, Virtual Machine Platform enabled

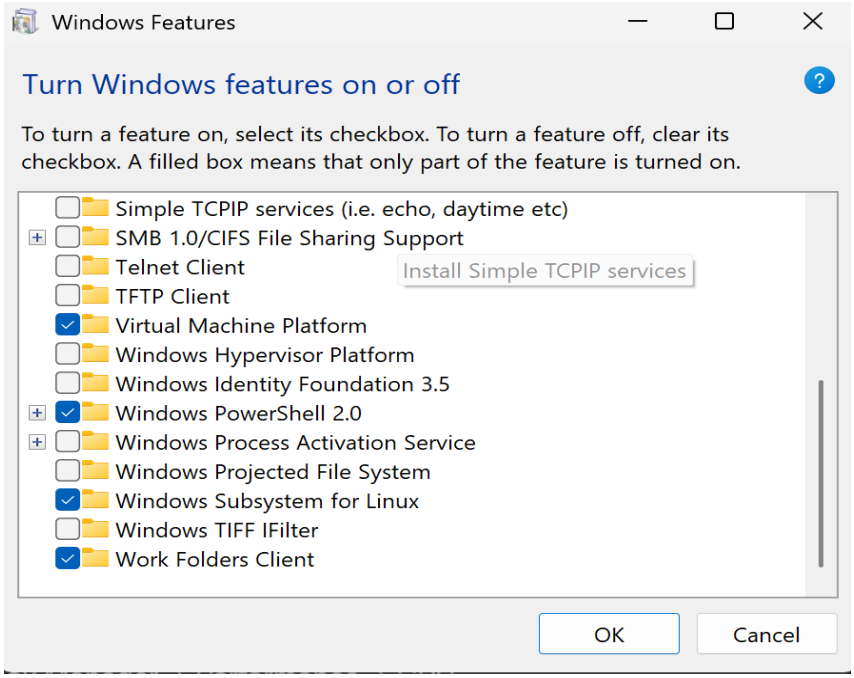

Figure 3: Windows Config for WSL

```
PS C:\Users\shrey> wsl -l -v
  NAME                   STATE           VERSION
* Ubuntu-24.04           Running         2
  Ubuntu                 Stopped         2
PS C:\Users\shrey>
```

Figure 4: WSL List

- **GPU Details:** Nvidia GeForce RTX 4060 laptop GPU

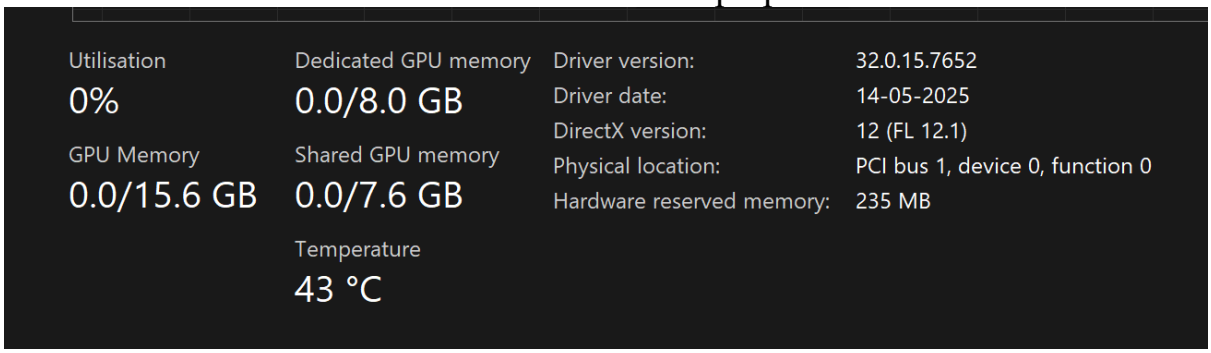

Figure 5: GPU Properties

- **WSL distro and kernel**

```
shreyas@Comet:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 24.04.2 LTS
Release:        24.04
Codename:       noble
shreyas@Comet:~$ uname -r
6.6.87.2-microsoft-standard-WSL2
shreyas@Comet:~$
```

Figure 6: WSL Info

- **Python, pip, git versions**

```
6.6.87.2-microsoft-standard-WSL2
shreyas@Comet:~$ python3 --version
Python 3.12.3
shreyas@Comet:~$ pip3 --version
pip 24.0 from /usr/lib/python3/dist-packages/pip (python 3.12)
shreyas@Comet:~$ git --version
git version 2.43.0
shreyas@Comet:~$
```

Figure 7: Installed Basic tools with version

- **NVIDIA GPU visibility in WSL**

```
shreyas@Comet:~$ nvidia-smi
Sat Aug  9 16:19:29 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.144.03             Driver Version: 576.52         CUDA Version: 12.9     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 4060 ...    On  |   00000000:01:00.0 Off |                  N/A |
| N/A   43C    P3             12W /   42W |       0MiB /   8188MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

Figure 8: Nvidia Properties in WSL

2 Tools Used

2.1 Version Control: Git: 2.43.0

```
shreyas@Comet:~$ git --version
git version 2.43.0
shreyas@Comet:~$
```

Figure 9: Git Version

2.2 PassGAN AI (AI Candidate Generation)

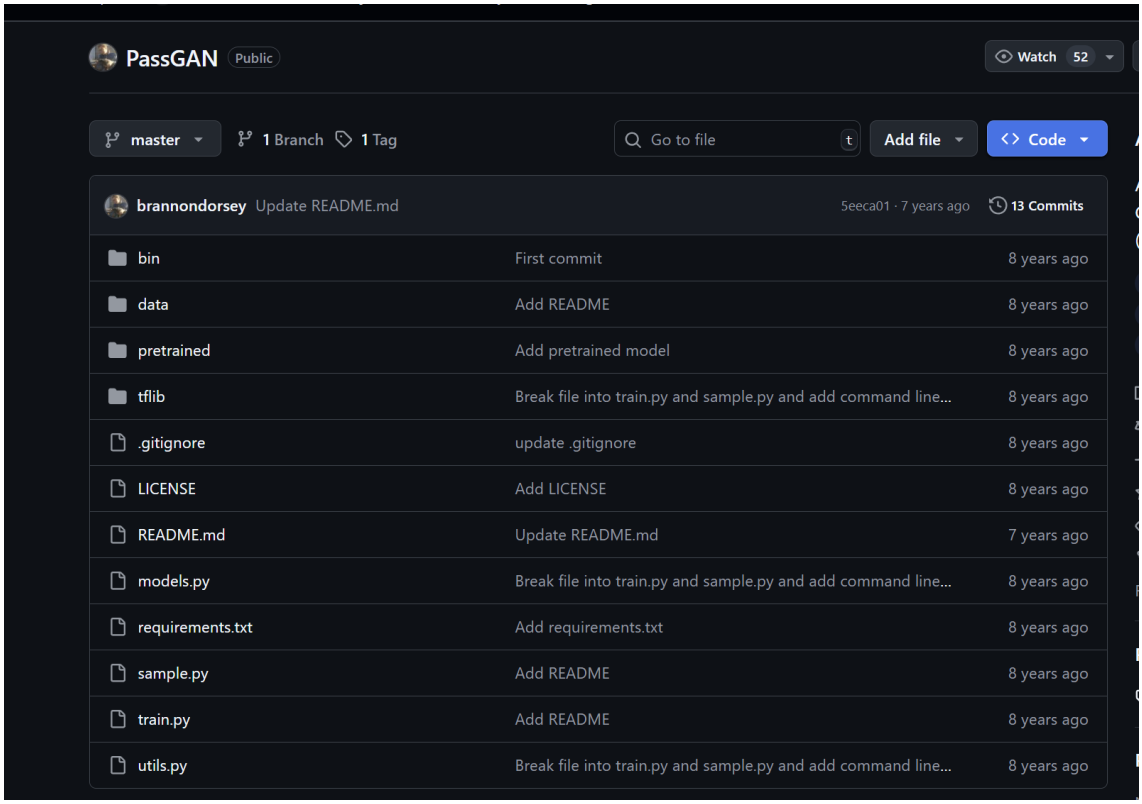

Figure 10: PassGAN repository (Link)

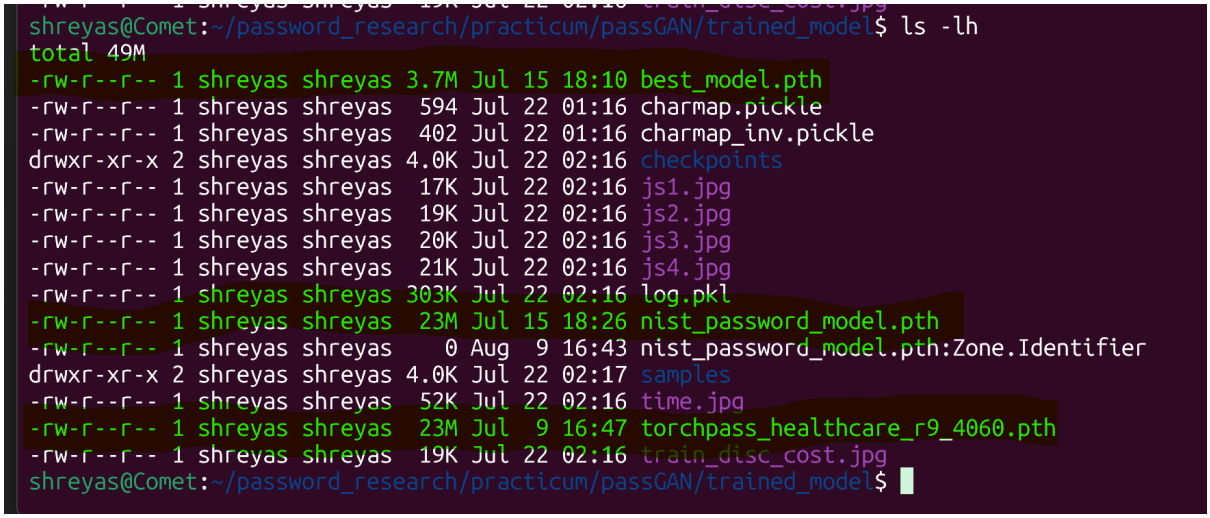

Figure 11: Models Trained using passGAN (Traditional/NIST/Healthcare)

- **Node – 23.11.1**

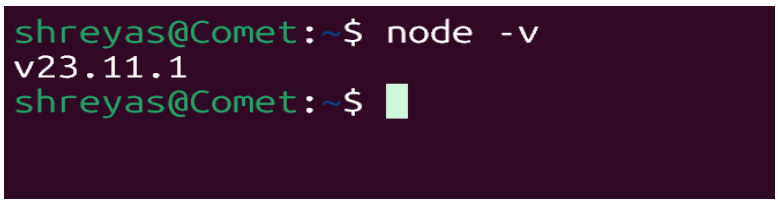

Figure 12: Node Version

- **Hashcat**

```
shreyas@Comet:~$ hashcat -I
hashcat (v6.2.6) starting in backend information mode

cuInit(): no CUDA-capable device is detected

OpenCL Info:
============

OpenCL Platform ID #1
  Vendor..: The pocl project
  Name....: Portable Computing Language
  Version.: OpenCL 3.0 PoCL 5.0+debian  Linux, None+Asserts, RELOC, SPIR, LLVM 16.0.6, SLEEF, DISTRO, POCL_DEBUG

  Backend Device ID #1
    Type...........: CPU
    Vendor.ID......: 1
    Vendor.........: AuthenticAMD
    Name...........: cpu-skylake-avx512-AMD Ryzen 9 8945HS w/ Radeon 780M Graphics
    Version........: OpenCL 3.0 PoCL HSTR: cpu-x86_64-pc-linux-gnu-skylake-avx512
    Processor(s)...: 16
    Clock..........: 3992
    Memory.Total...: 5534 MB (limited to 1024 MB allocatable in one block)
    Memory.Free....: 2735 MB
    Local.Memory...: 1024 KB
    OpenCL.Version.: OpenCL C 1.2 PoCL
    Driver.Version.: 5.0+debian
```

Figure 13: Hashcat Version Details

3 Python Scripts Used in the Project.

3.1 Combine and Filter Breach Lists

- **Purpose:** Create a deduplicated, normalized master list of common passwords, suitable for exclusion and for baseline rules attacks.

- **Inputs:**
  - data/raw/rockyou.txt
  - data/raw/pwdb_top_1m.txt
  - data/raw/xato-10m.txt

- **Outputs**
  - data/processed/common_passwords_dedup.txt
  - data/processed/common_passwords_stats.json

- **Key parameters**
  - Min/max length bounds
  - Unicode normalization and casing
  - Optional language-only filtering (ASCII or extended)

- **Validation**
  - Count >0; duplicates removed.
  - Spot-check a few known entries appear once.
  - JSON stats include total, unique, min/max length.

```python
import unicodedata, json
from pathlib import Path

IN_FILES = [
    Path("data/raw/rockyou.txt"),
    Path("data/raw/pwdb_top_1m.txt"),
    Path("data/raw/xato-10m.txt"),
]
OUT_TXT = Path("data/processed/common_passwords_dedup.txt")
OUT_JSON = Path("data/processed/common_passwords_stats.json")
MIN_LEN, MAX_LEN = 4, 128

def norm(s: str) -> str:
    s = s.strip()
    s = unicodedata.normalize("NFKC", s)
    return s

def main():
    seen = set()
    total_raw = 0
    for p in IN_FILES:
        with p.open("r", encoding="utf-8", errors="ignore") as f:
            for line in f:
                total_raw += 1
                s = norm(line)
                if not s:
                    continue
                if not (MIN_LEN <= len(s) <= MAX_LEN):
                    continue
                seen.add(s)
    OUT_TXT.parent.mkdir(parents=True, exist_ok=True)
    with OUT_TXT.open("w", encoding="utf-8") as f:
        for s in sorted(seen):
            f.write(s + "\n")
    stats = {
        "inputs": [str(p) for p in IN_FILES],
        "total_raw_lines": total_raw,
        "unique": len(seen),
        "min_len": MIN_LEN,
        "max_len": MAX_LEN,
        "output": str(OUT_TXT),
    }
    with OUT_JSON.open("w", encoding="utf-8") as f:
        json.dump(stats, f, indent=2)
    print(f"Wrote {len(seen)} unique passwords -> {OUT_TXT}")

if __name__ == "__main__":
    main()
```

Figure 14: Snippet to Combine & Filter Common Passwords

3.2 Fetch ICD-11 Terms and Filter Medical Vocabulary

- **Purpose**: Fetch ICD-11 content and extract a clean list of medical terms for the “medical-themed” distribution.
- **Inputs:** ICD-11 API/JSON dump or downloaded JSON (data/raw/icd11_full.json)
- **Outputs:**
  - data/processed/icd11_terms_raw.json
  - data/processed/medical_terms.txt

- **Key parameters:**
  - Language code (en)
  - Token length and character filters
  - De-duplication and lowercase normalization

- **Validation**
  - Non-empty term list; random spot-check for correctness.
  - No duplicates; reasonable token lengths.

```
# icd11_terms_min.py
import json,re
from pathlib import Path
SRC=Path("data/raw/icd11_full.json");
OUT=Path("data/processed/medical_terms.txt");
OUT.parent.mkdir(parents=True,exist_ok=True)
j=json.loads(SRC.read_text(encoding="utf-8")); terms=set()
def walk(o):
    if isinstance(o,dict):
        for k,v in o.items():
            if isinstance(v,str) and k.lower() in
{"title","label","name","definition"}:
                t=re.sub(r"\s+"," ",v.strip().lower())
                if 3<=len(t)<=64 and re.fullmatch(r"[a-z0-9\-
\'\s]+",t): terms.add(t)
            else: walk(v)
    elif isinstance(o,list):
        for x in o: walk(x)
walk(j)
OUT.write_text("\n".join(sorted(terms)),encoding="utf-8")
print(f"Extracted {len(terms)} terms -> {OUT}")
```

Figure 15: Extract medical terms from ICD-11 JSON

3.3 Generate synthetic passwords (NIST/Traditional/Medical)

- Inputs: data/processed/medical_terms.txt
- Outputs: data/processed/synth_{nist,traditional,medical}.txt

```
# generate_synth_min.py
import random,string
from pathlib import Path
random.seed(42)
MED=Path("data/processed/medical_terms.txt").read_text(encoding="utf-
8").splitlines()
OUT=Path("data/processed"); OUT.mkdir(parents=True,exist_ok=True)
def nist():
    words = MED if len(MED)>5000 else
["correct","battery","staple","horse","clinic","patient"]
    k=random.randint(3,5); return "-".join(random.sample(words,k))
def trad():
    w="".join(random.choice(string.ascii_letters) for _ in
range(random.randint(6,10)))
    return w.title()+ "".join(random.choice(string.digits) for _ in range(2,4))
+ random.choice("!@#$%^&*")
def med():
    base=random.choice(MED).replace(" ",""); return (base +
random.choice(["","123","!","@2024"]))[:64]
def write(pth, gen, n): pth.write_text("\n".join(gen() for _ in
range(n)),encoding="utf-8")
N=10_000_000; n_nist,n_trad,n_med = int(N*0.4), int(N*0.3), int(N*0.3)
write(OUT/"synth_nist.txt", nist, n_nist)
write(OUT/"synth_traditional.txt", trad, n_trad)
write(OUT/"synth_medical.txt", med, n_med)
print("Done.")
```

Figure 16: Generating Passwords of all 3 Categories using PassGAN models

3.4 Filter unusable passwords

- **Input:** any list (e.g., data/processed/synth_nist.txt).
- **Output:** data/processed/clean/<name>_clean.txt

```
# filter_passwords_min.py
import argparse,string
from pathlib import Path
ap=argparse.ArgumentParser(); ap.add_argument("--input",required=True);
ap.add_argument("--exclude",default="")
args=ap.parse_args()
src=Path(args.input); out=Path("data/processed/clean");
out.mkdir(parents=True,exist_ok=True)
excl=set(Path(args.exclude).read_text(encoding="utf-8").splitlines()) if
args.exclude else set()
PRINT=set(string.printable); MINL,MAXL=4,128
def ok(s): return s and MINL<=len(s)<=MAXL and set(s)<=PRINT and s not in excl
lines=[l.rstrip("\n") for l in src.open(encoding="utf-8",errors="ignore")]
kept=[s for s in lines if ok(s)]
(Path(out/f"{src.stem}_clean.txt")).write_text("\n".join(kept),encoding="utf-8")
print(f"Kept {len(kept)}/{len(lines)}")
```

Figure 17: Filtering Unusable Passwords Generated by Models

3.5 Split into train/eval and hash eval

- **Input:** concatenated clean list (e.g., data/processed/clean/all_clean.txt)
- **Outputs:** data/splits/{train.txt,eval_plain.txt,eval_hashes.txt}

```
# split_hash_min.py
import hashlib,random
from pathlib import Path
random.seed(2024)
IN=Path("data/processed/clean/all_clean.txt"); OUT=Path("data/splits");
OUT.mkdir(parents=True,exist_ok=True)
lines=[l.strip() for l in IN.read_text(encoding="utf-8").splitlines()
if l.strip()]
random.shuffle(lines); n=int(len(lines)*0.5)
train,evalp = lines[:n], lines[n:]
(OUT/"train.txt").write_text("\n".join(train),encoding="utf-8")
(OUT/"eval_plain.txt").write_text("\n".join(evalp),encoding="utf-8")
evalh=[hashlib.sha256(s.encode()).hexdigest() for s in evalp]
(OUT/"eval_hashes.txt").write_text("\n".join(evalh),encoding="utf-8")
print("Split+hash done.")
```

Figure 18: Splitting Filtered Dataset in 2 parts

3.6 Track crack times and build metrics CSV

- **Inputs:** eval_plain.txt, eval_hashes.txt, attacks/{rules_only,ai_only}/cracked.txt (format: hash TAB epoch_seconds)
- **Output:** analysis/outputs/metrics.csv

```
import csv,sys
from pathlib import Path
EPL=Path("data/splits/eval_plain.txt"); EHS=Path("data/splits/eval_hashes.txt")
RCR=Path("attacks/rules_only/cracked.txt");
ACR=Path("attacks/ai_only/cracked.txt")
START_RULES=0.0; START_AI=0.0  # set to epoch start times
OUT=Path("analysis/outputs"); OUT.mkdir(parents=True,exist_ok=True)
pl=EPL.read_text(encoding="utf-8").splitlines(); hs=EHS.read_text(encoding="utf-
8").splitlines()
h2p=dict(zip(hs,pl))
def load(p):
    d={};
    if not p.exists(): return d
    for line in p.read_text(encoding="utf-8",errors="ignore").splitlines():
        parts=line.split()
        if len(parts)>=2:
            try: d[parts[0]]=float(parts[1])
            except: pass
    return d
r,a=load(RCR),load(ACR)
with open(OUT/"metrics.csv","w",newline="",encoding="utf-8") as f:
    w=csv.writer(f);
w.writerow(["password","category","strength","length","complexity_score","crack_ti
me_hours_traditional","crack_time_hours_ai"])
    for h,pwd in h2p.items():
        cat="nist"  # replace if you have per-row categories
        length=len(pwd); strength=length; comp=sum(c.isdigit() for c in
pwd)+sum(not c.isalnum() for c in pwd)
        tr = ((r[h]-START_RULES)/3600.0) if h in r else ""
        ta = ((a[h]-START_AI)/3600.0) if h in a else ""
        w.writerow([pwd,cat,strength,length,comp,tr,ta])
print("metrics.csv written.")
```

Figure 19: Building CSV

4 Web Application: AI Password Analyzer

4.1 Key behaviors

- All checks run in-browser; no network calls by default.
- Feedback covers: length, character variety, patterns (years, repeats, keyboard runs).
- Optional policy mode selector (NIST/Traditional/Medical) for messaging.
- UI note: “All analysis runs locally in your browser. No passwords are sent to a server.”

4.2 Steps to run locally

Option A: Static server (recommended)

- git clone https://github.com/shreyasunhle0110/ai-password-analyzer
- cd ai-password-analyzer
- python -m http.server 8080
- Open http://localhost:8080

4.3 Security/CSP (short)

- Client-only processing; no password transmission.
- Recommended CSP: default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline'; img-src 'self' data: (via meta tag or vercel.json).

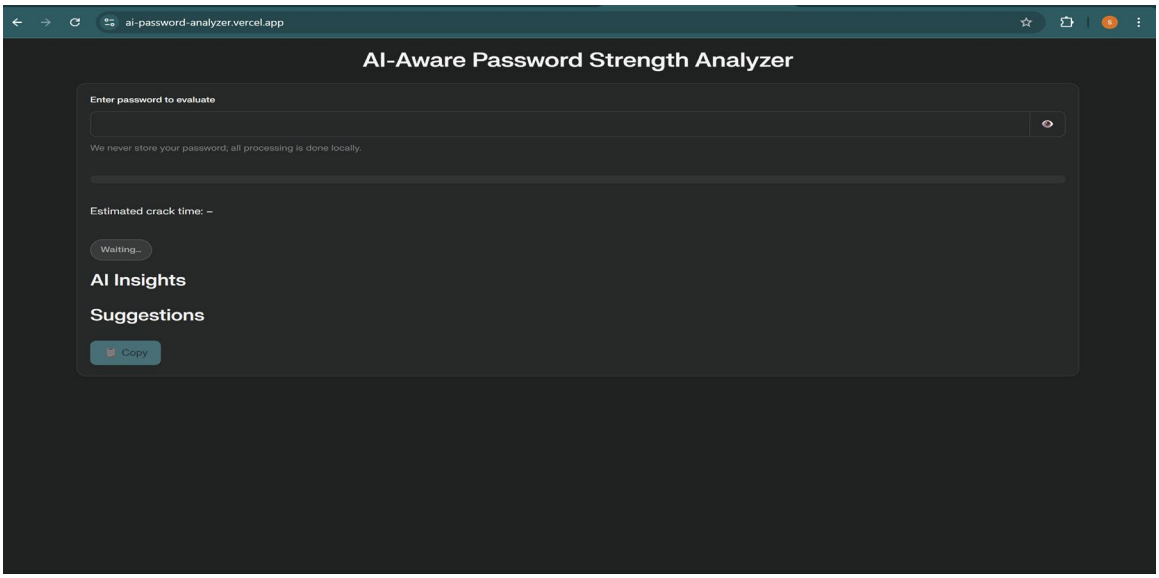

Figure 20: App home

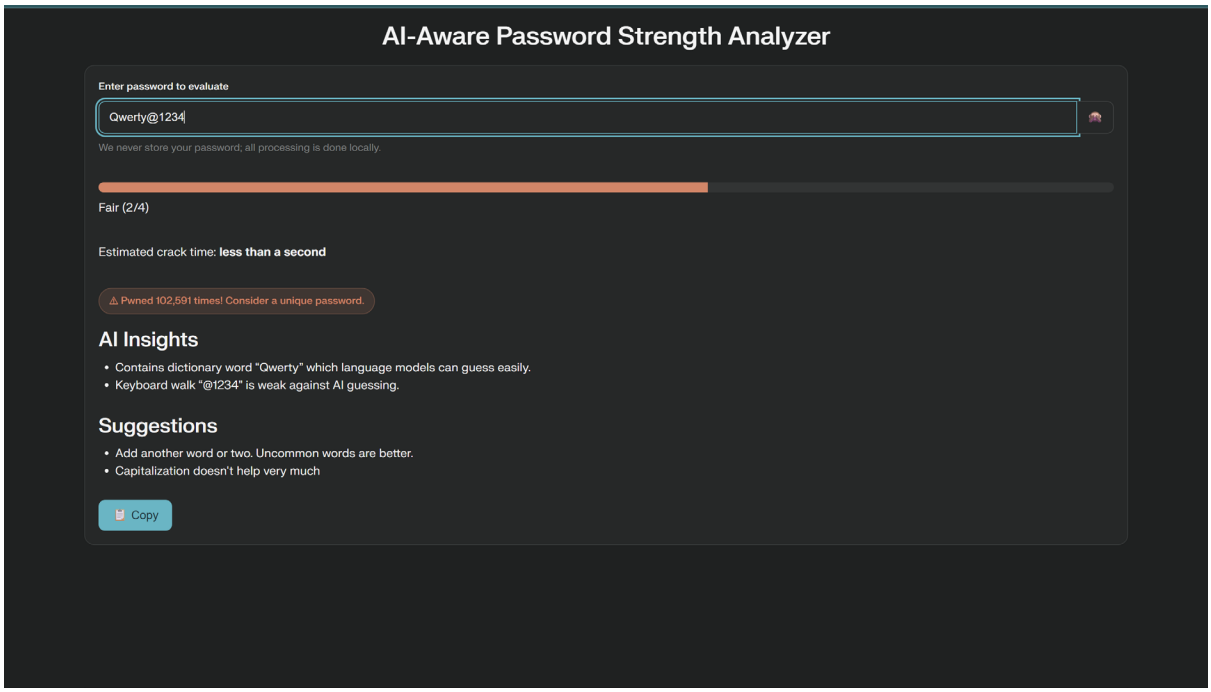

Figure 21: Feedback shown for weak example

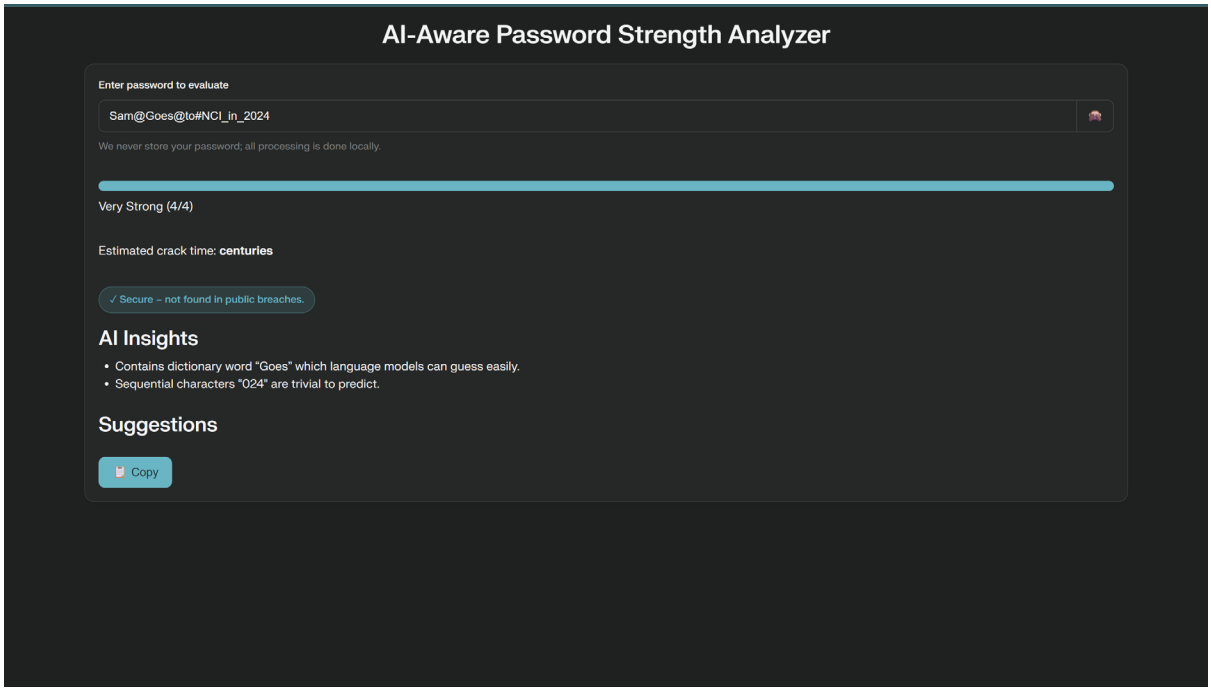

Figure 22: Feedback for strong passphrases