LAMPIRAN

Lampiran 1 Surat Keterangan Bebas Plagiat

UNIVERSITAS DARMA PERSADA
UPT PERPUSTAKAAN
Gedung Rektorat Lantai 3,
Jl.Taman Malaka Selatan, Pondok Kelapa – Jakarta Timur 13450

SURAT KETERANGAN
HASIL PENGECEKAN TURNITIN

UPT Perpustakaan Universitas Darma Persada menerangkan telah selesai melakukan pemeriksaan duplikasi/*similarity* menggunakan perangkat lunak Turnitin terhadap hasil karya sebagai berikut:

Judul	: PENGEMBANGAN SISTEM REKOMENDASI BERBASIS POPULARITAS DAN TREN INVESTASI CRYPTOCURRENCY DENGAN METODE: FEATURE-ENHANCED CF DAN NEURAL CF
Penulis	: Muhammad Faiz Aqil Fathoni
NIM	: **2021230006**
Tgl pemeriksaan	: 24 Juli 2025

Dengan hasil Tingkat Kesamaan (*similarity index*) **6%**

Demikian Surat Keterangan kami buat, untuk dipergunakan sebagaimana mestinya.

Jakarta, 22 Juli 2025

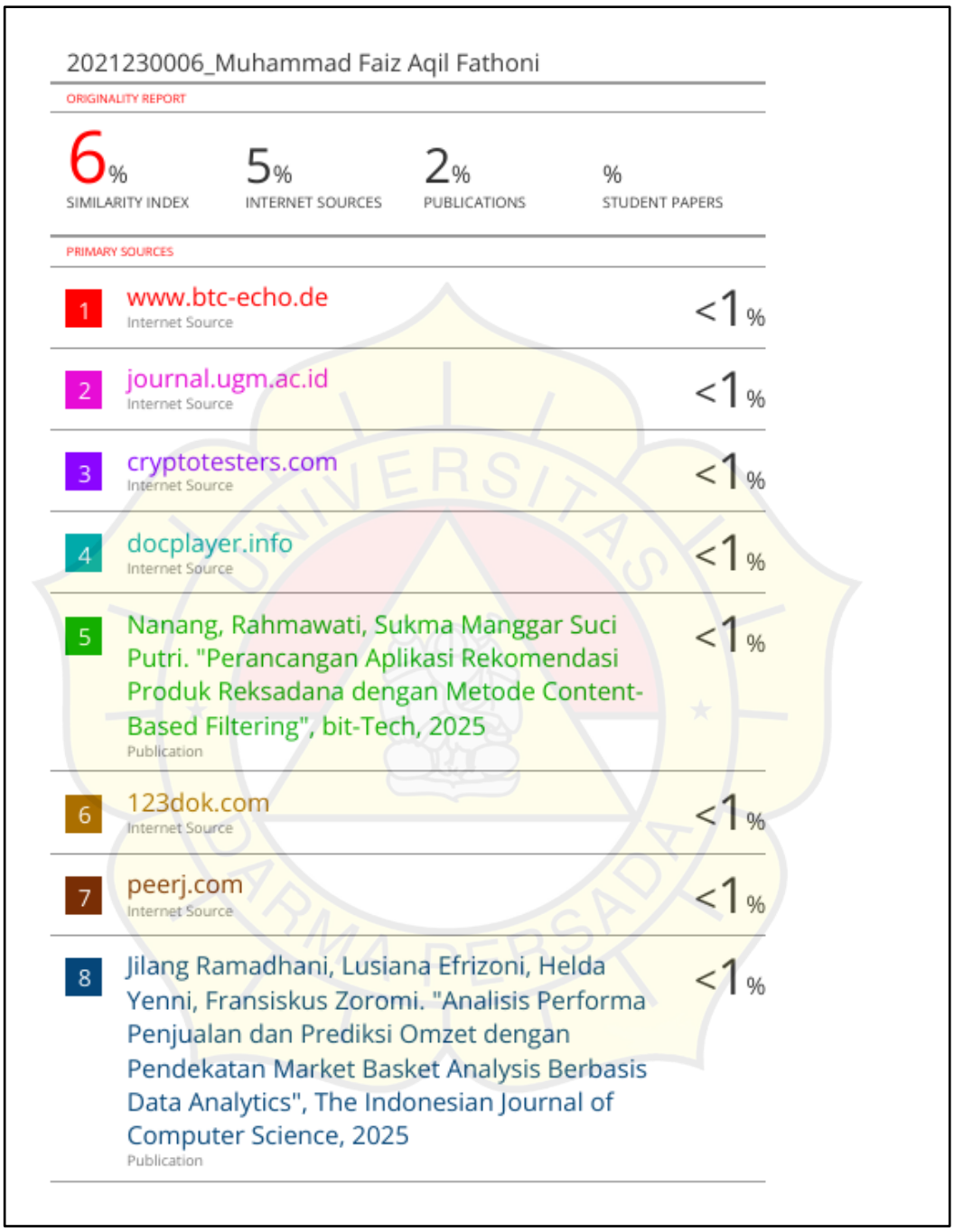
Ka.UPT Perpustakaan Unsada

Yus Rusmiyati, SS., MM

Batas maksimal similarity 30% untuk Fakultas Sastra dan Ekonomi

Batas maksimal similarity 25% untuk Fakultas Teknik, Kelautan dan Pasca Sarjana

Lampiran 2 Hasil Turnitin



2021230006_Muhammad Faiz Aqil Fathoni

ORIGINALITY REPORT

6%	5%	2%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.btc-echo.de Internet Source	<1%
2	journal.ugm.ac.id Internet Source	<1%
3	cryptotesters.com Internet Source	<1%
4	docplayer.info Internet Source	<1%
5	Nanang, Rahmawati, Sukma Manggar Suci Putri. "Perancangan Aplikasi Rekomendasi Produk Reksadana dengan Metode Content-Based Filtering", bit-Tech, 2025 Publication	<1%
6	123dok.com Internet Source	<1%
7	peerj.com Internet Source	<1%
8	Jilang Ramadhani, Lusiana Efrizoni, Helda Yenni, Fransiskus Zoromi. "Analisis Performa Penjualan dan Prediksi Omzet dengan Pendekatan Market Basket Analysis Berbasis Data Analytics", The Indonesian Journal of Computer Science, 2025 Publication	<1%

Lampiran 3 Source Code

```
# ============================================
# file: alt_fecf.py
# ============================================

import json
import logging
import math
import pickle
from dataclasses import dataclass, asdict
from datetime import datetime, timezone
from pathlib import Path
from typing import Any, Dict, Iterable, List, Literal, Optional,
Sequence, Tuple

import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import StandardScaler

# ------------------------------------------------
# Logging
# ------------------------------------------------
logger = logging.getLogger(__name__)
if not logger.handlers:
    logging.basicConfig(
        level=logging.INFO,
        format="%(asctime)s %(levelname)s [%(name)s] %(message)s"
    )

# ------------------------------------------------
# Metrik evaluasi (implicit feedback)
# ------------------------------------------------
def precision_at_k(actual: Sequence[Any], predicted:
Sequence[Any], k: int) -> float:
    """Precision@k."""
    if k <= 0 or not predicted:
        return 0.0
    topk = list(predicted)[:k]
    hits = sum(1 for x in topk if x in set(actual))
    return hits / k


def recall_at_k(actual: Sequence[Any], predicted: Sequence[Any],
k: int) -> float:
    """Recall@k."""
    if not actual:
        return 0.0
    topk = list(predicted)[:k]
    hits = sum(1 for x in topk if x in set(actual))
    return hits / len(set(actual))
```

```
def dcg_at_k(rels: Sequence[int]) -> float:
    """DCG dasar untuk daftar relevansi biner/real."""
    return sum((rel / math.log2(i + 2)) for i, rel in
enumerate(rels))


def ndcg_at_k(actual: Sequence[Any], predicted: Sequence[Any], k:
int) -> float:
    """nDCG@k (biner)."""
    topk = list(predicted)[:k]
    rels = [1 if x in set(actual) else 0 for x in topk]
    idcg = dcg_at_k(sorted(rels, reverse=True))
    if idcg == 0.0:
        return 0.0
    return dcg_at_k(rels) / idcg


def hit_rate_at_k(actual: Sequence[Any], predicted: Sequence[Any],
k: int) -> float:
    """HitRate@k."""
    topk = list(predicted)[:k]
    return 1.0 if any(x in set(actual) for x in topk) else 0.0


# ------------------------------------------------
# Konfigurasi & util
# ------------------------------------------------
@dataclass
class FECFConfig:
    n_components: int = 64                  # rank SVD
    alpha_content: float = 0.4              # bobot konten vs CF
(0..1)
    half_life_days: float = 30.0            # tren (semakin kecil
→ lebih responsif)
    max_neighbors: int = 200                # item-item
neighborhood dipakai saat scoring
    user_col: str = "user_id"
    item_col: str = "item_id"
    weight_col: str = "weight"              # bobot interaksi
(implicit), default=1
    timestamp_col: Optional[str] = "timestamp"  # epoch detik / ms
/ ISO8601
    feature_cols: Optional[List[str]] = None # kolom fitur item
(numerik atau kategori)

    def to_json(self) -> str:
        return json.dumps(asdict(self), ensure_ascii=False,
indent=2)


def _to_epoch_seconds(ts: Any) -> Optional[float]:
    """Konversi berbagai format timestamp → epoch detik."""
    if ts is None or (isinstance(ts, float) and math.isnan(ts)):
        return None
    if isinstance(ts, (int, float)):
        # heuristik ms vs s
```

```
        return float(ts) / 1000.0 if ts > 1e12 else float(ts)
    if isinstance(ts, str):
        try:
            return datetime.fromisoformat(ts.replace("Z",
"+00:00")).timestamp()
        except Exception:
            return None
    if isinstance(ts, datetime):
        if ts.tzinfo is None:
            ts = ts.replace(tzinfo=timezone.utc)
        return ts.timestamp()
    return None


def _apply_time_decay(w: float, t: Optional[float], now:
Optional[float], half_life_days: float) -> float:
    """Peluruhan eksponensial pada bobot interaksi untuk
memodelkan tren (mengutamakan interaksi terbaru)."""
    if now is None or t is None or half_life_days <= 0:
        return float(w)
    hl_seconds = half_life_days * 86400.0
    decay = 0.5 ** ((now - t) / hl_seconds)
    return float(w) * float(decay)


def _ensure_categorical(df: pd.DataFrame, cols: List[str]) ->
pd.DataFrame:
    """One-hot untuk kolom kategorikal; kolom numerik dilewatkan
apa adanya."""
    cat_cols, num_cols = [], []
    for c in cols:
        if pd.api.types.is_numeric_dtype(df[c]):
            num_cols.append(c)
        else:
            cat_cols.append(c)
    out = []
    if num_cols:
        out.append(df[num_cols].astype(float))
    if cat_cols:
        out.append(pd.get_dummies(df[cat_cols].astype("category"),
dummy_na=False))
    return pd.concat(out, axis=1) if out else
pd.DataFrame(index=df.index)


# ------------------------------------------------
# Model FECF
# ------------------------------------------------
class FeatureEnhancedCF:
    """
    Feature-Enhanced Collaborative Filtering (FECF).

    Pipeline ringkas:
    1) bentuk matriks user-item (csr), terapkan time-decay ke
bobot.
```

```
    2) SVD pada matriks → faktor laten item; cosine-similarity
(CF).
    3) siapkan matriks fitur item (one-hot + standar) → cosine-
similarity (konten).
    4) blend similarity: S = (1-α)*S_cf + α*S_content.
    5) rekomendasi: agregasi similarity tetangga item dari histori
pengguna.

    Kenapa time-decay? Mengikat topik "tren" agar interaksi
terbaru lebih berpengaruh.
    """

    def __init__(self, config: Optional[FECFConfig] = None) ->
None:
        self.config = config or FECFConfig()
        self._fitted = False

        # Artefak
        self.user_index: Dict[Any, int] = {}
        self.item_index: Dict[Any, int] = {}
        self.index_user: List[Any] = []
        self.index_item: List[Any] = []

        self.ui_matrix: Optional[csr_matrix] = None
        self.item_factors: Optional[np.ndarray] = None
        self.svd: Optional[TruncatedSVD] = None

        self.item_features: Optional[pd.DataFrame] = None
        self.scaler: Optional[StandardScaler] = None

        self.sim_cf: Optional[np.ndarray] = None
        self.sim_content: Optional[np.ndarray] = None
        self.sim_blend: Optional[np.ndarray] = None

        # meta
        self._seen_by_user: Dict[int, set] = {}

    # ------------------------------
    # Fitting
    # ------------------------------
    def fit(
        self,
        interactions: pd.DataFrame,
        items: pd.DataFrame,
        *,
        user_col: Optional[str] = None,
        item_col: Optional[str] = None,
        weight_col: Optional[str] = None,
        timestamp_col: Optional[str] = None,
        feature_cols: Optional[List[str]] = None,
    ) -> "FeatureEnhancedCF":
        """
        Latih model FECF.

        Parameters
        ----------
```

```
        interactions : DataFrame[user,item,weight,timestamp?]
        items        : DataFrame[item, fitur...]
        user_col     : nama kolom user
        item_col     : nama kolom item
        weight_col   : nama kolom bobot (default=1 jika tidak ada)
        timestamp_col: cap waktu interaksi (opsional, untuk time-
decay)
        feature_cols : daftar kolom fitur item yang dipakai
(default: semua kecuali id)
        """
        cfg = self.config
        ucol = user_col or cfg.user_col
        icol = item_col or cfg.item_col
        wcol = weight_col or cfg.weight_col
        tcol = timestamp_col if timestamp_col is not None else
cfg.timestamp_col

        assert ucol in interactions.columns and icol in
interactions.columns, "Kolom user/item tidak ditemukan"
        if wcol not in interactions.columns:
            interactions = interactions.copy()
            interactions[wcol] = 1.0

        # time-decay
        now =
interactions[tcol].dropna().apply(_to_epoch_seconds).max() if
(tcol and tcol in interactions) else None
        if tcol and tcol in interactions.columns:
            ts = interactions[tcol].apply(_to_epoch_seconds)
        else:
            ts = pd.Series([None] * len(interactions),
index=interactions.index)
        interactions["_w"] = [
            _apply_time_decay(w, t, now, cfg.half_life_days) for
w, t in zip(interactions[wcol].values, ts.values)
        ]

        # reindex pengguna & item (idx kontigu)
        self.index_user =
interactions[ucol].astype("category").cat.categories.tolist()
        self.index_item = pd.concat([interactions[icol],
items[icol]]).astype("category").cat.categories.tolist()
        self.user_index = {u: i for i, u in
enumerate(self.index_user)}
        self.item_index = {i: j for j, i in
enumerate(self.index_item)}

        # sparse csr
        row = interactions[ucol].map(self.user_index).to_numpy()
        col = interactions[icol].map(self.item_index).to_numpy()
        data = interactions["_w"].astype(float).to_numpy()
        n_users = len(self.user_index)
        n_items = len(self.item_index)
        self.ui_matrix = csr_matrix((data, (row, col)),
shape=(n_users, n_items))
```

```
        # objek 'seen' per user untuk filter rekomendasi
        self._seen_by_user = {}
        for u, i in zip(row, col):
            self._seen_by_user.setdefault(int(u),
set()).add(int(i))

        # SVD → faktor item
        self.svd = TruncatedSVD(n_components=cfg.n_components,
random_state=42)
        # SVD pada matriks item-user (kolom = user) stabil untuk
cosine antar item
        iu = self.ui_matrix.T  # shape: (items, users)
        logger.info("Fitting TruncatedSVD...")
        self.item_factors = self.svd.fit_transform(iu)  #
(n_items, k)

        # similarity CF (cosine antar item)
        logger.info("Computing item-item similarity (CF)...")
        self.sim_cf = cosine_similarity(self.item_factors)  #
(n_items, n_items)
        np.fill_diagonal(self.sim_cf, 0.0)

        # siapkan fitur item
        if feature_cols is None:
            feature_cols = [c for c in items.columns if c != icol]
        feats =
_ensure_categorical(items.set_index(icol).reindex(self.index_item)
, feature_cols)
        if feats.shape[1] == 0:
            self.sim_content = np.zeros((n_items, n_items),
dtype=float)
        else:
            self.scaler = StandardScaler(with_mean=False)  #
dengan sparse compat
            X =
self.scaler.fit_transform(feats.values.astype(float))
            logger.info("Computing item-item similarity
(content)...")
            self.sim_content = cosine_similarity(X)
            np.fill_diagonal(self.sim_content, 0.0)

        # blending
        a = float(cfg.alpha_content)
        self.sim_blend = (1.0 - a) * self.sim_cf + a *
self.sim_content

        self._fitted = True
        logger.info("FECF fitted: users=%d, items=%d, k=%d",
n_users, n_items, cfg.n_components)
        return self

    # ----------------------------
    # Inferensi rekomendasi
    # ----------------------------
    def _score_user(self, user_idx: int) -> np.ndarray:
```

```
        """Skor semua item untuk seorang user dengan neighborhood
similarity."""
        assert self._fitted and self.sim_blend is not None and
self.ui_matrix is not None
        seen = self._seen_by_user.get(user_idx, set())
        # prefer only neighbors of items seen by user (hemat &
fokus)
        if not seen:
            # cold-start user → gunakan popularitas global (norm
row-sum similarity)
            pop = np.asarray(self.ui_matrix.sum(axis=0)).ravel()
            return pop / (pop.max() + 1e-8)
        sim = self.sim_blend  # (I, I)
        scores = np.zeros(sim.shape[0], dtype=float)
        # agregasi similarity dari item histori
        for it in seen:
            # ambil tetangga teratas untuk kecepatan
            neigh_idx = np.argpartition(-sim[it],
kth=min(sim.shape[0]-1, self.config.max_neighbors))[:
self.config.max_neighbors]
            scores[neigh_idx] += sim[it, neigh_idx]
        # nolkan item yang sudah dilihat
        if seen:
            scores[list(seen)] = -np.inf
        return scores

    def recommend(
        self,
        user_id: Any,
        k: int = 10,
        return_scores: bool = False
    ) -> List[Any] | List[Tuple[Any, float]]:
        """Top-k rekomendasi untuk satu user."""
        if not self._fitted:
            raise RuntimeError("Model belum di-fit.")
        if user_id not in self.user_index:
            # cold-start: user baru
            scores =
np.asarray(self.ui_matrix.sum(axis=0)).ravel()
            idx = np.argpartition(-scores, kth=min(k, len(scores)-
1))[:k]
            items = [self.index_item[i] for i in idx[np.argsort(-
scores[idx])]]
            return items if not return_scores else list(zip(items,
scores[idx[np.argsort(-scores[idx])]]))

        uidx = self.user_index[user_id]
        scores = self._score_user(uidx)
        top_idx = np.argpartition(-scores, kth=min(k, len(scores)-
1))[:k]
        order = top_idx[np.argsort(-scores[top_idx])]
        result_items = [self.index_item[i] for i in order]
        return result_items if not return_scores else
list(zip(result_items, scores[order].tolist()))

    # ----------------------------
```

```
    # Evaluasi
    # ----------------------------
    def evaluate(
        self,
        test_interactions: pd.DataFrame,
        k: int = 10,
        user_col: Optional[str] = None,
        item_col: Optional[str] = None
    ) -> Dict[str, float]:
        """
        Evaluasi sederhana pada set uji (implicit).
        Gunakan leave-one-out/holdout di luar fungsi ini untuk
pembagian data.
        """
        if not self._fitted:
            raise RuntimeError("Model belum di-fit.")
        ucol = user_col or self.config.user_col
        icol = item_col or self.config.item_col

        grouped =
test_interactions.groupby(ucol)[icol].apply(list)
        precs, recs, ndcgs, hits = [], [], [], []
        for uid, actual_items in grouped.items():
            preds = self.recommend(uid, k=k, return_scores=False)
            precs.append(precision_at_k(actual_items, preds, k))
            recs.append(recall_at_k(actual_items, preds, k))
            ndcgs.append(ndcg_at_k(actual_items, preds, k))
            hits.append(hit_rate_at_k(actual_items, preds, k))
        return {
            f"precision@{k}": float(np.mean(precs) if precs else
0.0),
            f"recall@{k}": float(np.mean(recs) if recs else 0.0),
            f"ndcg@{k}": float(np.mean(ndcgs) if ndcgs else 0.0),
            f"hit_rate@{k}": float(np.mean(hits) if hits else
0.0),
        }

    # ----------------------------
    # Persistensi
    # ----------------------------
    def save(self, path: str | Path) -> None:
        """Simpan model dan artefak ke file .pkl."""
        if not self._fitted:
            raise RuntimeError("Model belum di-fit.")
        path = Path(path)
        obj = {
            "config": asdict(self.config),
            "index_user": self.index_user,
            "index_item": self.index_item,
            "item_factors": self.item_factors,
            "sim_cf": self.sim_cf,
            "sim_content": self.sim_content,
            "sim_blend": self.sim_blend,
            "seen_by_user": {int(k): list(v) for k, v in
self._seen_by_user.items()},
            "scaler_mean": getattr(self.scaler, "mean_", None),
```

```
            "scaler_scale": getattr(self.scaler, "scale_", None),
        }
        with path.open("wb") as f:
            pickle.dump(obj, f)
        logger.info("Model tersimpan di %s", str(path))

    @classmethod
    def load(cls, path: str | Path) -> "FeatureEnhancedCF":
        """Muat model dari file .pkl."""
        path = Path(path)
        with path.open("rb") as f:
            obj = pickle.load(f)
        model = cls(FECFConfig(**obj["config"]))
        model.index_user = obj["index_user"]
        model.index_item = obj["index_item"]
        model.user_index = {u: i for i, u in
enumerate(model.index_user)}
        model.item_index = {it: j for j, it in
enumerate(model.index_item)}
        model.item_factors = obj["item_factors"]
        model.sim_cf = obj["sim_cf"]
        model.sim_content = obj["sim_content"]
        model.sim_blend = obj["sim_blend"]
        model._seen_by_user = {int(k): set(map(int, v)) for k, v
in obj["seen_by_user"].items()}
        # scaler optional
        if obj.get("scaler_scale") is not None:
            model.scaler = StandardScaler(with_mean=False)
            model.scaler.scale_ = np.asarray(obj["scaler_scale"])
            model.scaler.mean_ = np.asarray(obj["scaler_mean"]) if
obj.get("scaler_mean") is not None else None
        model._fitted = True
        logger.info("Model dimuat dari %s", str(path))
        return model
```

```
# ============================================
# file: ncf.py
# ============================================

import logging
import math
import pickle
from dataclasses import dataclass, asdict
from pathlib import Path
from typing import Any, Dict, Iterable, List, Optional, Sequence,
Tuple

import numpy as np
import pandas as pd
import torch
from torch import nn
from torch.utils.data import DataLoader, Dataset

# ------------------------------------------------
# Logging
```

```
# ------------------------------------------------
logger = logging.getLogger(__name__)
if not logger.handlers:
    logging.basicConfig(
        level=logging.INFO,
        format="%(asctime)s %(levelname)s [%(name)s] %(message)s"
    )

# ------------------------------------------------
# Metrik (sama dengan FECF untuk konsistensi)
# ------------------------------------------------
def precision_at_k(actual: Sequence[Any], predicted:
Sequence[Any], k: int) -> float:
    if k <= 0 or not predicted:
        return 0.0
    topk = list(predicted)[:k]
    hits = sum(1 for x in topk if x in set(actual))
    return hits / k


def recall_at_k(actual: Sequence[Any], predicted: Sequence[Any],
k: int) -> float:
    if not actual:
        return 0.0
    topk = list(predicted)[:k]
    hits = sum(1 for x in topk if x in set(actual))
    return hits / len(set(actual))


def dcg_at_k(rels: Sequence[int]) -> float:
    return sum((rel / math.log2(i + 2)) for i, rel in
enumerate(rels))


def ndcg_at_k(actual: Sequence[Any], predicted: Sequence[Any], k:
int) -> float:
    topk = list(predicted)[:k]
    rels = [1 if x in set(actual) else 0 for x in topk]
    idcg = dcg_at_k(sorted(rels, reverse=True))
    if idcg == 0.0:
        return 0.0
    return dcg_at_k(rels) / idcg


def hit_rate_at_k(actual: Sequence[Any], predicted: Sequence[Any],
k: int) -> float:
    topk = list(predicted)[:k]
    return 1.0 if any(x in set(actual) for x in topk) else 0.0


# ------------------------------------------------
# Dataset implicit dengan negative sampling
# ------------------------------------------------
@dataclass
class NCFConfig:
    user_col: str = "user_id"
```

```
    item_col: str = "item_id"
    weight_col: str = "weight"
    timestamp_col: Optional[str] = "timestamp"

    n_users: Optional[int] = None
    n_items: Optional[int] = None

    embedding_dim: int = 64
    mlp_layers: Tuple[int, ...] = (128, 64)
    dropout: float = 0.2

    lr: float = 1e-3
    batch_size: int = 2048
    epochs: int = 10
    negative_per_positive: int = 4
    half_life_days: float = 30.0  # untuk time-decay pada bobot
contoh (opsional)

    seed: int = 42

    def to_dict(self) -> Dict[str, Any]:
        return asdict(self)


def _to_epoch_seconds(ts: Any) -> Optional[float]:
    if ts is None or (isinstance(ts, float) and math.isnan(ts)):
        return None
    if isinstance(ts, (int, float)):
        return float(ts) / 1000.0 if ts > 1e12 else float(ts)
    if isinstance(ts, str):
        try:
            from datetime import datetime, timezone
            return datetime.fromisoformat(ts.replace("Z",
"+00:00")).timestamp()
        except Exception:
            return None
    if hasattr(ts, "timestamp"):
        return float(ts.timestamp())
    return None


def _time_decay(w: float, t: Optional[float], now:
Optional[float], half_life_days: float) -> float:
    if now is None or t is None or half_life_days <= 0:
        return float(w)
    hl = half_life_days * 86400.0
    return float(w) * (0.5 ** ((now - t) / hl))


class ImplicitNCFDataset(Dataset):
    """
    Dataset implicit feedback untuk NCF dengan negative sampling.
    Sampling negatif dilakukan per-epoch agar bervariasi.
    """

    def __init__(
```

```
        self,
        interactions: pd.DataFrame,
        *,
        cfg: NCFConfig,
    ) -> None:
        ucol, icol, wcol, tcol = cfg.user_col, cfg.item_col,
cfg.weight_col, cfg.timestamp_col
        assert ucol in interactions and icol in interactions,
"Kolom user/item tidak ditemukan"

        df = interactions[[ucol, icol] + ([wcol] if wcol in
interactions else []) + ([tcol] if (tcol and tcol in interactions)
else [])].copy()
        if wcol not in df:
            df[wcol] = 1.0

        # reindex kontigu
        self.users =
df[ucol].astype("category").cat.categories.tolist()
        self.items =
df[icol].astype("category").cat.categories.tolist()
        self.u2i = {u: i for i, u in enumerate(self.users)}
        self.i2i = {it: j for j, it in enumerate(self.items)}
        self.n_users = len(self.users)
        self.n_items = len(self.items)

        # mapping ke index
        df["u"] = df[ucol].map(self.u2i).astype(np.int64)
        df["i"] = df[icol].map(self.i2i).astype(np.int64)

        # time-decay bobot contoh
        now = df[tcol].dropna().apply(_to_epoch_seconds).max() if
(tcol and tcol in df) else None
        ts = df[tcol].apply(_to_epoch_seconds) if (tcol and tcol
in df) else pd.Series([None] * len(df), index=df.index)
        df["_w"] = [_time_decay(w, t, now, cfg.half_life_days) for
w, t in zip(df[wcol].values, ts.values)]

        # set positif per user
        self.positives_by_user: Dict[int, set] = {}
        for u, it in zip(df["u"].values, df["i"].values):
            self.positives_by_user.setdefault(int(u),
set()).add(int(it))

        self.df = df[["u", "i", "_w"]].reset_index(drop=True)
        self.neg_k = int(cfg.negative_per_positive)
        self.rng = np.random.default_rng(cfg.seed)

        # cache untuk rekomendasi (filter histori)
        self.seen_by_user = {u: set(items) for u, items in
self.positives_by_user.items()}

    def __len__(self) -> int:
        return len(self.df) * (1 + self.neg_k)
```

```
    def __getitem__(self, idx: int) -> Tuple[torch.Tensor,
torch.Tensor, torch.Tensor]:
        # sampling dinamis: mapping idx → apakah negatif/positif
        pos_len = len(self.df)
        if idx < pos_len:
            row = self.df.iloc[idx]
            return (
                torch.tensor(row["u"], dtype=torch.long),
                torch.tensor(row["i"], dtype=torch.long),
                torch.tensor(1.0, dtype=torch.float32),
            )
        # sample negatif
        pos_idx = idx % pos_len
        row = self.df.iloc[pos_idx]
        u = int(row["u"])
        # pilih item negatif yang belum pernah dilihat
        while True:
            it = int(self.rng.integers(0, len(self.items)))
            if it not in self.positives_by_user.get(u, set()):
                break
        return (
            torch.tensor(u, dtype=torch.long),
            torch.tensor(it, dtype=torch.long),
            torch.tensor(0.0, dtype=torch.float32),
        )


# ------------------------------------------------
# Model Neural CF (GMF + MLP)
# ------------------------------------------------
class NeuralCF(nn.Module):
    """
    Neural Collaborative Filtering:
    - GMF branch: elemen-wise product embedding user & item.
    - MLP branch: concatenation → MLP.
    - Fusion: penjumlahan logit GMF + MLP → sigmoid.
    """

    def __init__(self, n_users: int, n_items: int, embedding_dim:
int, mlp_layers: Sequence[int], dropout: float) -> None:
        super().__init__()
        self.user_gmf = nn.Embedding(n_users, embedding_dim)
        self.item_gmf = nn.Embedding(n_items, embedding_dim)

        self.user_mlp = nn.Embedding(n_users, embedding_dim)
        self.item_mlp = nn.Embedding(n_items, embedding_dim)

        mlp: List[nn.Module] = []
        in_dim = 2 * embedding_dim
        for h in mlp_layers:
            mlp += [nn.Linear(in_dim, h), nn.ReLU(inplace=True),
nn.Dropout(p=dropout)]
            in_dim = h
        self.mlp = nn.Sequential(*mlp) if mlp else nn.Identity()

        self.fc_gmf = nn.Linear(embedding_dim, 1, bias=False)
```

```
        self.fc_mlp = nn.Linear(in_dim, 1, bias=True)

        self._init_weights()

    def _init_weights(self) -> None:
        # init menjaga stabilitas pelatihan
        for emb in [self.user_gmf, self.item_gmf, self.user_mlp,
self.item_mlp]:
            nn.init.normal_(emb.weight, std=0.01)
        for m in self.modules():
            if isinstance(m, nn.Linear):
                nn.init.xavier_uniform_(m.weight)
                if m.bias is not None:
                    nn.init.zeros_(m.bias)

    def forward(self, u: torch.Tensor, i: torch.Tensor) ->
torch.Tensor:
        # GMF
        ug, ig = self.user_gmf(u), self.item_gmf(i)
        gmf = ug * ig
        # MLP
        um, im = self.user_mlp(u), self.item_mlp(i)
        mlp_in = torch.cat([um, im], dim=-1)
        mlp_out = self.mlp(mlp_in)
        # fusion
        logit = self.fc_gmf(gmf) + self.fc_mlp(mlp_out)
        return torch.sigmoid(logit).squeeze(-1)


# ---------------------------------------------------
# Pelatihan & inferensi
# ---------------------------------------------------
@dataclass
class TrainState:
    cfg: NCFConfig
    n_users: int
    n_items: int
    model: NeuralCF


def train_ncf(
    interactions: pd.DataFrame,
    cfg: Optional[NCFConfig] = None,
    *,
    device: Optional[str] = None,
) -> TrainState:
    """
    Latih NCF dari DataFrame interaksi implicit.

    Parameters
    ----------
    interactions : DataFrame[user_id, item_id, weight?,
timestamp?]
    cfg          : NCFConfig
    device       : "cuda" / "cpu" (default: auto)
    """
```

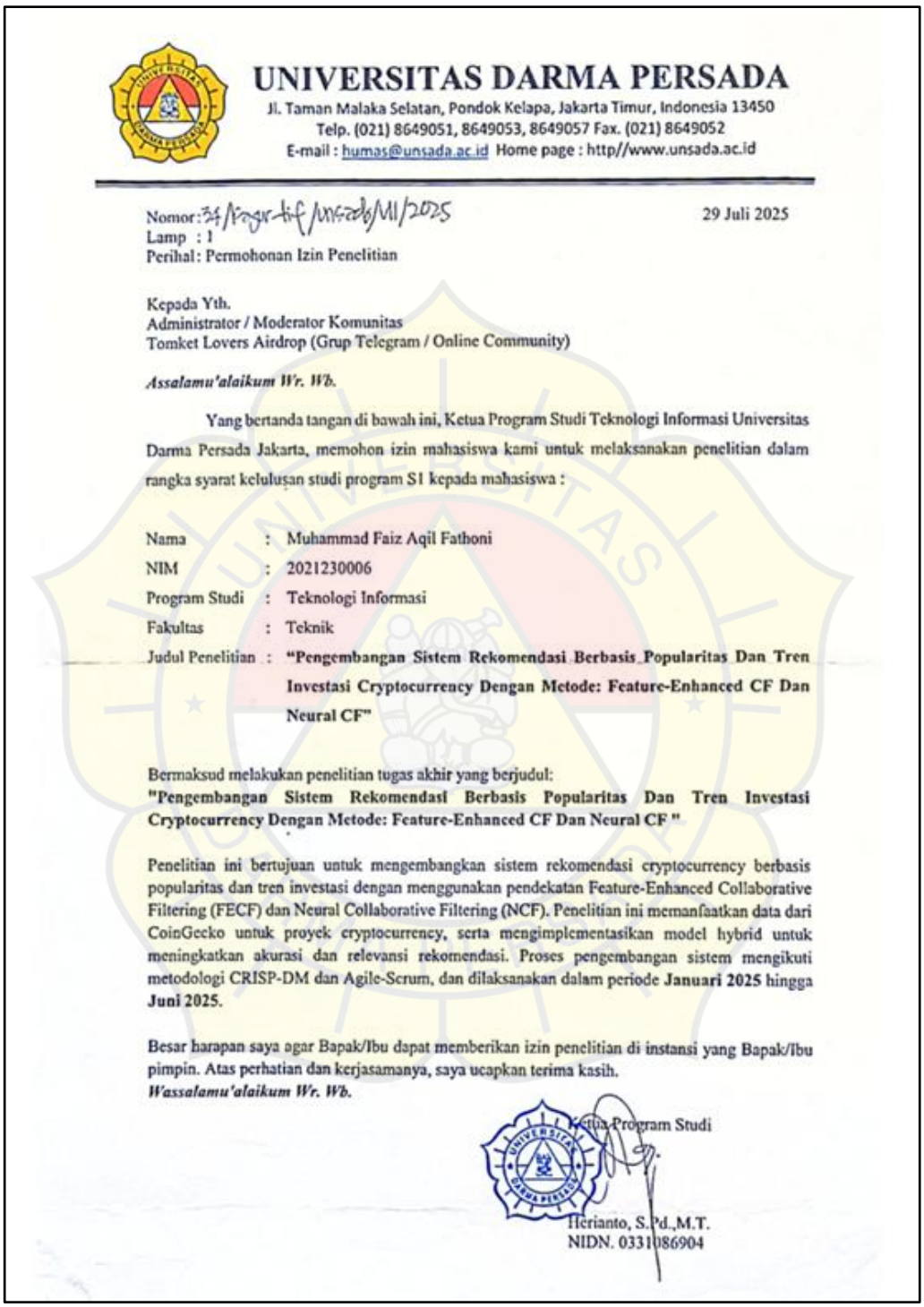
```
    cfg = cfg or NCFConfig()
    ds = ImplicitNCFDataset(interactions, cfg=cfg)
    n_users, n_items = ds.n_users, ds.n_items

    model = NeuralCF(
        n_users=n_users,
        n_items=n_items,
        embedding_dim=cfg.embedding_dim,
        mlp_layers=cfg.mlp_layers,
        dropout=cfg.dropout,
    )
    device = device or ("cuda" if torch.cuda.is_available() else
"cpu")
    model.to(device)

    optim = torch.optim.Adam(model.parameters(), lr=cfg.lr)
    criterion = nn.BCELoss()

    loader = DataLoader(ds, batch_size=cfg.batch_size,
shuffle=True, num_workers=0, drop_last=False)

    logger.info("Training NCF: users=%d items=%d epochs=%d",
n_users, n_items, cfg.epochs)
    model.train()
    for epoch in range(1, cfg.epochs + 1):
        running = 0.0
        for u, it, y in loader:
            u = u.to(device)
            it = it.to(device)
            y = y.to(device)

            optim.zero_grad(set_to_none=True)
            pred = model(u, it)
            loss = criterion(pred, y)
            loss.backward()
            optim.step()
            running += float(loss.detach().cpu().item()) * len(u)
        logger.info("Epoch %d/%d - loss=%.5f", epoch, cfg.epochs,
running / len(ds))

    return TrainState(cfg=cfg, n_users=n_users, n_items=n_items,
model=model)


def _rank_items_for_user(
    state: TrainState,
    user_idx: int,
    seen_items: Optional[set],
    top_k: int,
    batch: int = 4096,
) -> List[int]:
    """Skor seluruh item untuk satu user secara batch
(efisien)."""
    model = state.model
    device = next(model.parameters()).device
    model.eval()
```

```
    all_items = np.arange(state.n_items, dtype=np.int64)
    scores = np.empty_like(all_items, dtype=np.float32)

    with torch.no_grad():
        start = 0
        while start < len(all_items):
            end = min(start + batch, len(all_items))
            u = torch.full((end - start,), user_idx,
dtype=torch.long, device=device)
            it = torch.tensor(all_items[start:end],
dtype=torch.long, device=device)
            scores[start:end] = model(u,
it).detach().cpu().numpy()
            start = end

    if seen_items:
        scores[list(seen_items)] = -np.inf  # hindari item yang
sudah pernah dilihat

    top_idx = np.argpartition(-scores, kth=min(top_k, len(scores)-
1))[:top_k]
    order = top_idx[np.argsort(-scores[top_idx])]
    return order.tolist()


def recommend(
    state: TrainState,
    interactions_fit: pd.DataFrame,
    user_id: Any,
    k: int = 10,
) -> List[Any]:
    """
    Rekomendasi top-k untuk satu user.

    Catatan: butuh `interactions_fit` (yang dipakai saat training)
untuk memetakan
    user/item asli → index internal & untuk mengetahui histori
(filter).
    """
    ucol, icol = state.cfg.user_col, state.cfg.item_col

    # bangun kembali kategori untuk mapping (stabil terhadap
urutan)
    users =
interactions_fit[ucol].astype("category").cat.categories.tolist()
    items =
interactions_fit[icol].astype("category").cat.categories.tolist()
    u2i = {u: i for i, u in enumerate(users)}
    i2i = {it: j for j, it in enumerate(items)}

    if user_id not in u2i:
        # cold-start → populer global (frekuensi item)
        pop = interactions_fit[icol].value_counts()
        return pop.index[:k].tolist()

    user_idx = u2i[user_id]
```

```
    seen = set(
        interactions_fit.loc[interactions_fit[ucol] == user_id,
icol].map(i2i).dropna().astype(int).tolist()
    )
    item_idx = _rank_items_for_user(state, user_idx,
seen_items=seen, top_k=k)
    inv_item = {j: it for it, j in i2i.items()}
    return [inv_item[j] for j in item_idx]


def evaluate(
    state: TrainState,
    interactions_fit: pd.DataFrame,
    interactions_test: pd.DataFrame,
    k: int = 10,
) -> Dict[str, float]:
    """
    Evaluasi sederhana pada set uji (implicit).
    """
    ucol, icol = state.cfg.user_col, state.cfg.item_col
    grouped = interactions_test.groupby(ucol)[icol].apply(list)

    precs, recs, ndcgs, hits = [], [], [], []
    for uid, actual_items in grouped.items():
        preds = recommend(state, interactions_fit, uid, k=k)
        precs.append(precision_at_k(actual_items, preds, k))
        recs.append(recall_at_k(actual_items, preds, k))
        ndcgs.append(ndcg_at_k(actual_items, preds, k))
        hits.append(hit_rate_at_k(actual_items, preds, k))
    return {
        f"precision@{k}": float(np.mean(precs) if precs else 0.0),
        f"recall@{k}": float(np.mean(recs) if recs else 0.0),
        f"ndcg@{k}": float(np.mean(ndcgs) if ndcgs else 0.0),
        f"hit_rate@{k}": float(np.mean(hits) if hits else 0.0),
    }


# ------------------------------------------------
# Persistensi
# ------------------------------------------------
def save(state: TrainState, path: str | Path) -> None:
    """Simpan state model ke .pkl (termasuk arsitektur &
bobot)."""
    path = Path(path)
    obj = {
        "cfg": state.cfg.to_dict(),
        "n_users": state.n_users,
        "n_items": state.n_items,
        "state_dict": state.model.state_dict(),
    }
    with path.open("wb") as f:
        pickle.dump(obj, f)
    logger.info("NCF tersimpan di %s", str(path))
```

```
def load(path: str | Path, device: Optional[str] = None) ->
TrainState:
    """Muat state model dari .pkl."""
    path = Path(path)
    with path.open("rb") as f:
        obj = pickle.load(f)
    cfg = NCFConfig(**obj["cfg"])
    model = NeuralCF(
        n_users=obj["n_users"],
        n_items=obj["n_items"],
        embedding_dim=cfg.embedding_dim,
        mlp_layers=cfg.mlp_layers,
        dropout=cfg.dropout,
    )
    device = device or ("cuda" if torch.cuda.is_available() else
"cpu")
    model.load_state_dict(obj["state_dict"])
    model.to(device)
    return TrainState(cfg=cfg, n_users=obj["n_users"],
n_items=obj["n_items"], model=model)
```

Lampiran 4 Permohonan Izin Penelitian (Unsada)



UNIVERSITAS DARMA PERSADA
Jl. Taman Malaka Selatan, Pondok Kelapa, Jakarta Timur, Indonesia 13450
Telp. (021) 8649051, 8649053, 8649057 Fax. (021) 8649052
E-mail : humas@unsada.ac.id Home page : http//www.unsada.ac.id

Nomor : 34/Pengr.tif/Unsada/VII/2025 — 29 Juli 2025
Lamp : 1
Perihal : Permohonan Izin Penelitian

Kepada Yth.
Administrator / Moderator Komunitas
Tomket Lovers Airdrop (Grup Telegram / Online Community)

Assalamu'alaikum Wr. Wb.

Yang bertanda tangan di bawah ini, Ketua Program Studi Teknologi Informasi Universitas Darma Persada Jakarta, memohon izin mahasiswa kami untuk melaksanakan penelitian dalam rangka syarat kelulusan studi program S1 kepada mahasiswa :

Nama	:	Muhammad Faiz Aqil Fathoni
NIM	:	2021230006
Program Studi	:	Teknologi Informasi
Fakultas	:	Teknik
Judul Penelitian	:	**"Pengembangan Sistem Rekomendasi Berbasis Popularitas Dan Tren Investasi Cryptocurrency Dengan Metode: Feature-Enhanced CF Dan Neural CF"**

Bermaksud melakukan penelitian tugas akhir yang berjudul:
"Pengembangan Sistem Rekomendasi Berbasis Popularitas Dan Tren Investasi Cryptocurrency Dengan Metode: Feature-Enhanced CF Dan Neural CF "

Penelitian ini bertujuan untuk mengembangkan sistem rekomendasi cryptocurrency berbasis popularitas dan tren investasi dengan menggunakan pendekatan Feature-Enhanced Collaborative Filtering (FECF) dan Neural Collaborative Filtering (NCF). Penelitian ini memanfaatkan data dari CoinGecko untuk proyek cryptocurrency, serta mengimplementasikan model hybrid untuk meningkatkan akurasi dan relevansi rekomendasi. Proses pengembangan sistem mengikuti metodologi CRISP-DM dan Agile-Scrum, dan dilaksanakan dalam periode **Januari 2025** hingga **Juni 2025**.

Besar harapan saya agar Bapak/Ibu dapat memberikan izin penelitian di instansi yang Bapak/Ibu pimpin. Atas perhatian dan kerjasamanya, saya ucapkan terima kasih.
Wassalamu'alaikum Wr. Wb.

Ketua Program Studi

Herianto, S.Pd.,M.T.
NIDN. 0331086904