ДОДАТОК А

Вихідний код

bit\main.py

```
import argparse import logging
from pathlib import Path
from fetch_data_legacy import get_candles from backtest import run_backtest
from strategies.rsi_macd import strategy_rsi_macd
def configure_logging(debug: bool):
level = logging.DEBUG if debug else logging.INFO logging.basicConfig(
format='[%(levelname)s] %(message)s', level=level
)
def parse_args():
parser = argparse.ArgumentParser(description=" Пісочниця стратегії BTC RSI+MACD")
parser.add_argument('--pair', default='BTC/USDT',
help='Торгова пара, наприклад
BTC/USDT')
parser.add_argument('--tf', '--timeframe', default='5m',
help='«Молодший» таймфрейм: 5m') parser.add_argument('--strategy', default='rsi_macd',
help='Назва стратегії (файл у strategies/ без .py)')
parser.add_argument('--debug', action='store_true',
help='Увімкнути докладний лог
(DEBUG)')
return parser.parse_args()
def main():
args = parse_args() configure_logging(args.debug)
if args.tf != '5m':
raise ValueError("Стратегія rsi_macd розрахована лише на 5m + H1-фільтр.")

# 1) H1 – для фільтрації тренду
```

```
    logging.info(" Завантажуємо історію %s (1h)...", args.pair)
    df_h1 = get_candles(args.pair, '1h', lookback=8760)
    # 2) 5m – для генерації сигналів
    logging.info(" Завантажуємо історію %s (5m)...", args.pair)
    df_5m = get_candles(args.pair, '5m', lookback=105120)
    # 3) Побудова сигналів по RSI+MACD
    logging.info(" Створюємо сигнали для стратегії %s
(TF=5m)...", args.strategy)
    if args.strategy == 'rsi_macd':
    entry_signal, exit_signal, stop_price, tp_price =
strategy_rsi_macd(df_5m, df_h1)
    else:
    raise ValueError(f"Стратегію {args.strategy} не
    знайдено.")
    # 4) Запускаємо бектест logging.info(" Запускаємо бектест")
metrics, equity_curve = run_backtest(
    df_5m, entry_signal, exit_signal, stop_price,
    tp_price, risk_per_trade=0.02, init_cash=1000,
slippage=0.0005, fee=0.0005
    )
    # 5) Виводимо метрики logging.info(" Готово! Метрики:")
    logging.info(" Річний прибут %0.2f
                                    ок: %%",
    metrics['annual_retu * 100)
rn']
    logging.info(" • Частка прибуткових угод:
%0.2f%%", metrics['win_rate'] * 100)
    logging.info(" • Максимальна просадка:
%0.2f%%", metrics['max_drawdown'] * 100)
    logging.info(" • Шарп-співвідношення: %0.2f",
metrics['sharpe'])
    logging.info(" • Фактор прибутку: %0.2f",
metrics['profit_factor'])
    # 6) Зберігаємо криву капіталу
Path('logs').mkdir(exist_ok=True)
```

```
    equity_curve.to_csv(f"logs/equity_{args.strategy}_{args.tf}.
cs v")
    logging.info("    Крива    капіталу    збережена    в
logs/equity_%s_%s.csv", args.strategy, args.tf)
    if _name_== "_main_": main()
```

bit\indicators.py

```
    import pandas as pd import numpy as np
    def  ema(series:  pd.Series,  window:  int)  ->  pd.Series:
return series.ewm(span=window, adjust=False).mean()
    def atr(high: pd.Series, low: pd.Series, close: pd.Series,
window: int = 14) -> pd.Series:
    high_low = high - low
    high_close = (high - close.shift(1)).abs() low_close = (low
- close.shift(1)).abs()
    tr   =   pd.concat([high_low,   high_close,   low_close],
axis=1).max(axis=1)
    return tr.rolling(window=window, min_periods=1).mean() def
rsi(series: pd.Series, window: int = 14) -> pd.Series:
    delta = series.diff()
    gain = delta.clip(lower=0) loss = -delta.clip(upper=0)
    avg_gain            =            gain.rolling(window=window,
min_periods=window).mean()
    avg_loss            =            loss.rolling(window=window,
min_periods=window).mean()
    rs = avg_gain / avg_loss return 100 - (100 / (1 + rs))
    def macd(series: pd.Series, fast: int = 12, slow: int = 26,
signal: int = 9):
    ema_fast = series.ewm(span=fast, adjust=False).mean()
    ema_slow   =   series.ewm(span=slow,   adjust=False).mean()
macd_line = ema_fast - ema_slow
    signal_line          =          macd_line.ewm(span=signal,
adjust=False).mean()
```

```
    histogram = macd_line - signal_line return macd_line,
signal_line, histogram
    def hma(series: pd.Series, window: int) -> pd.Series:
    half_length = int(window / 2) sqrt_length =
int(np.sqrt(window))
    ema_half =
                            series.ewm(span=half_length,
adjust=False).mean()
    ema_full =
                             series.ewm(span=window,
adjust=False).mean()
    diff = (2 * ema_half) - ema_full
    hma_series =
                            diff.ewm(span=sqrt_length,
adjust=False).mean()
    return hma_series
```

bit\fetch_data_legacy.py

```
    import ccxt
    import pandas as pd from pathlib import Path import time
    try:
    from config import BINANCE_API_KEY, BINANCE_SECRET
    binance_params = {
    'apiKey': BINANCE_API_KEY, 'secret': BINANCE_SECRET,
    'enableRateLimit': True
    }
    except ImportError:
    binance_params = {'enableRateLimit': True} BINANCE =
ccxt.binance(binance_params)
    def _fetch_ohlcv_full(symbol: str, timeframe: str, since:
int, limit_per_call: int = 1000) -> pd.DataFrame:
    ohlcv_all = []
    timeframe_ms = BINANCE.parse_timeframe(timeframe) * 1000
    fetch_since = since while True:
```

```
    ohlcv = BINANCE.fetch_ohlcv(symbol, timeframe=timeframe,
since=fetch_since, limit=limit_per_call)
    if not ohlcv: break
    ohlcv_all.extend(ohlcv)
    last_ts = ohlcv[-1][0]
    fetch_since = last_ts + timeframe_ms
    if len(ohlcv) < limit_per_call: break
    time.sleep(BINANCE.rateLimit / 1000)
    df = pd.DataFrame(ohlcv_all, columns=['time', 'open',
    'high', 'low', 'close', 'volume'])
    df['time'] = pd.to_datetime(df['time'], unit='ms')
df.drop_duplicates(subset='time', keep='last',
    inplace=True)
    df.set_index('time', inplace=True)
df.sort_index(inplace=True) return df
    def download_candles(symbol: str, timeframe: str, since: int
= None, limit: int = 1000) -> pd.DataFrame:
    if since is None:
    # Получаем только последние limit баров ohlcv =
BINANCE.fetch_ohlcv(symbol,
    timeframe=timeframe, limit=limit)
    df = pd.DataFrame(ohlcv, columns=['time', 'open', 'high',
'low', 'close', 'volume'])
    df['time'] = pd.to_datetime(df['time'], unit='ms')
df.set_index('time', inplace=True)
    return df else:
    return _fetch_ohlcv_full(symbol, timeframe, since,
limit_per_call=1000)
    def get_candles(symbol: str, timeframe: str, lookback: int =
1000) -> pd.DataFrame:
    Path('data').mkdir(exist_ok=True) filename =
f"data/{symbol.replace('/',
    '_')}_{timeframe}.parquet" file_path = Path(filename) if
file_path.exists():
    return pd.read_parquet(file_path)
    now_ms = int(time.time() * 1000)
```

```
    timeframe_ms = BINANCE.parse_timeframe(timeframe) * 1000
since = now_ms - lookback * timeframe_ms
    df = download_candles(symbol, timeframe, since=since,
limit=lookback)
    df.to_parquet(file_path) return df
```

bit/config.py

```
    # config.py
    API_KEY=
    "ktyocuYAVVpsk6yxkK5eFkml5AH8ZaB5qY6DrqFBYfJKkvPurBgglFvBCr2
HQ z3r"
    API_SECRET                                                =
"****************igtXQgSbZKLRMB0l0rwHDM1Ccu2yu"
```

bit/ backtest_simple.py

```
    from typing import Any import pandas as pd
    def   run_backtest(df,   entry_long,   exit_long,
          entry_short, exit_short,
    sl_pct=0.01,tp_pct=0.03, initial_equity=10000, fee=0.0006,
    log_file=None, equity_curve_file=None, position_size=None):
    df = df.copy()  df['position'] = 0
    df['equity'] = initial_equity df['returns'] = 0.0
    equity = initial_equity position = 0
    entry_price = 0
    size_pct = 1.0 trades = []
    for i in range(1, len(df)): idx = df.index[i]
    price = df.loc[idx, 'close']
    prev_equity = df.loc[df.index[i - 1], 'equity'] if position
!= 0:
    stop = entry_price * (1 - sl_pct) if position == 1 else
entry_price * (1 + sl_pct)
    take = entry_price * (1 + tp_pct) if position == 1 else
entry_price * (1 - tp_pct)
```

```
    exit_cond = (
    (price <= stop or price >= take) if position ==
    1 else
    (price >= stop or price <= take)
    )
      forced_exit = (exit_long if position == 1 else
exit_short).iloc[i]

    if exit_cond or forced_exit:
    exit_type = 'long' if position == 1 else 'short' trades[-
1].update({'exit_time':idx,
    'exit_price': price})
    gross_return = (price - entry_price) / entry_price if
position == 1 else (entry_price - price) / entry_price
    net_return = gross_return - 2 * fee trade_profit =
net_return * equity * size_pct equity += trade_profit
    position = 0
    'short'
    if position == 0:
    if entry_long.iloc[i] or entry_short.iloc[i]: entry_price =
price
    position = 1 if entry_long.iloc[i] else -1 strat_type =
'long' if position == 1 else
    size_pct = position_size[idx] if position_size
    is not None else 1.0
    trades.append({
    'type': strat_type, 'entry_time': idx, 'entry_price':
price, 'position_pct': size_pct
    })
    df.loc[idx, 'position'] = position df.loc[idx, 'equity'] =
equity
    if log_file:
    pd.DataFrame(trades).to_csv(log_file, index=False) if
equity_curve_file:
    df[['equity']].to_csv(equity_curve_file) return df
```

```
    def calc_trade_profit(entry_price,       exit_price,
is_long, fee=0.0006):
    if is_long:
    gross_return  =  (exit_price  -  entry_price)  /  entry_price
else:
    gross_return  =  (entry_price  -  exit_price)  /  entry_price
net_return = gross_return - 2 * fee
    return net_return
```

bit/backtest.py

```
    import vectorbt as vbt import pandas as pd from typing import
Tuple

    def run_backtest(df: pd.DataFrame,
    entry_signal:    pd.Series,    exit_signal:    pd.Series,
stop_price:  pd.Series,  tp_price:  pd.Series,  risk_per_trade:
float = 0.05, init_cash: float = 1000, slippage: float = 0.0005,
    fee:  float  =  0.0005)  ->  Tuple[dict,
    pd.Series]:

    close = df['close'] high = df['high'] low = df['low']

    size = pd.Series(0.0, index=df.index)

    for i in df.index[entry_signal]: entry_price = close.loc[i]
sl_price = stop_price.loc[i]

    if sl_price == 0: size.loc[i] = 0.0 continue
    risk_per_unit = abs(entry_price - sl_price)
    risk_per_unit
    position_size = (init_cash * risk_per_trade) / size.loc[i]
= position_size
    pf     =     vbt.Portfolio.from_signals(     close=close,
entries=entry_signal,       exits=exit_signal,       size=size,
```

```
init_cash=init_cash,                freq=df.index.inferred_freq,
slippage=slippage, fees=fee,
     direction='both', call_seq='auto'
     )
     stats = pf.stats() 100 / 100 / N) - 1
     if 'Annualized Return [%]' in stats.index: annual_return =
stats['Annualized Return [%]'] /
     else:
     total_return_pct = stats.get('Total Return [%]', 0)
     try:
     N = len(df)
     annual_return = (1 + total_return_pct) ** (252



     except:
     annual_return = total_return_pct



     win_rate = stats.get('Win Rate [%]', 0) / 100 max_dd =
stats.get('Max Drawdown [%]', 0) / 100 sharpe = stats.get('Sharpe
Ratio', 0) profit_factor = stats.get('Profit Factor', 0)

     metrics = {
     'annual_return': annual_return, 'win_rate': win_rate,
'max_drawdown': max_dd, 'sharpe': sharpe, 'profit_factor':
profit_factor
     }
     equity_curve = pf.value() return metrics, equity_curve
```

bit/strategies/ fakey_hybrid.py

```
     import pandas as pd import numpy as np import ta
     import matplotlib.pyplot as plt import os

     BASE_LOG_PATH=
```

```
     os.path.abspath(os.path.join(os.path.dirname(_file_),"..",
"logs"))
     os.makedirs(BASE_LOG_PATH, exist_ok=True)

     def load_data():
     df=
     pd.read_parquet(os.path.join(os.path.dirname(_file_
),

"..", "data", "BTC_USDT_1h.parquet"))
     df.index = pd.to_datetime(df.index) return df

     def apply_indicators(df):
     df['ema50'] = ta.trend.ema_indicator(df['close'], 50) 200)



     14).rsi()
     df['ema200']   =    ta.trend.ema_indicator(df['close'],
     df['rsi']   =
            ta.momentum.RSIIndicator(df['close'], df['cci']=
            ta.trend.cci(df['high'],        df['low'],

     df['close'], 20)
     df['adx']            =   ta.trend.adx(df['high'],
                          df['low'], df['close'], 14)
     df['atr']=
     ta.volatility.average_true_range(df['high'],df['low'],
df['close'], 14)
     df['macd_hist']     =     ta.trend.macd_diff(df['close'])
df['volume_ma'] = df['volume'].rolling(20).mean() return df

     def detect_fakey(df):
     inside            =   (df['high']   <
                       df['high'].shift(1))  &
     (df['low'] > df['low'].shift(1))
```

```
    fakeout            =   (df['high']   >
                       df['high'].shift(2)) &
    (df['close'] < df['high'].shift(1)) return inside.shift(1)
& fakeout

    def generate_signals(df):
    df = apply_indicators(df) fakey = detect_fakey(df)
    atr_median = df['atr'].rolling(100).median()

    raw_long = (
    fakey & (df['ema50'] > df['ema200']) & (df['volume'] >
df['volume_ma']) & (df['atr'] >
    atr_median)
    )

    raw_short = (
              fakey & (df['ema50'] < df['ema200']) &
    (df['volume'] > df['volume_ma']) & (df['atr'] >
    atr_median)
    )

    entry_long            =   raw_long   &   (df['rsi']   >
                          55) & (df['macd_hist'] > 0)
    entry_short   =   raw_short   &   (df['rsi']   <   45)   &
(df['macd_hist'] < 0)

    exit_long = df['rsi'] < 50 exit_short = df['rsi'] > 50

    return entry_long, exit_long, entry_short, exit_short,
    df


    def run_backtest(df, entry_long, exit_long, entry_short,
exit_short, rr=3.5):
    df = df.copy() df['position'] = 0
    df['equity'] = 10000 trades = [] position = 0
```

```
    sl = tp = None


    for i in range(3, len(df)): price = df['close'].iloc[i]

    if position != 0:
    if position == 1 and (price <= sl or price >= tp or
exit_long.iloc[i]):
    trades[-1].update({'exit_time': df.index[i], 'exit_price':
price, 'result': 'TP/SL/Exit'})
    position = 0
    elif position == -1 and (price >= sl or price
    <= tp or exit_short.iloc[i]):
    trades[-1].update({'exit_time': df.index[i], 'exit_price':
price, 'result': 'TP/SL/Exit'})
    position = 0
    df.at[df.index[i], 'position'] = position continue

    if entry_long.iloc[i]: entry = price
    sl_base = df['low'].iloc[i - 1]
    sl = min(sl_base, entry - df['atr'].iloc[i] *
    0.8)
    risk = entry - sl
    tp = entry + risk * rr
    trades.append({'type': 'long', 'entry_time':
    df.index[i], 'entry_price': entry, 'sl': sl, 'tp': tp})
    position = 1


    elif entry_short.iloc[i]: entry = price
    sl_base = df['high'].iloc[i - 1]
    sl = max(sl_base, entry + df['atr'].iloc[i] *
    0.8)
    risk = sl - entry
    tp = entry - risk * rr
    trades.append({'type': 'short', 'entry_time':
```

```
            df.index[i], 'entry_price': entry, 'sl': sl, 'tp': tp})
        position = -1

                        df.at[df.index[i], 'position'] = position

        df['position']          =          df['position'].fillna(0)
df['returns']                   =     df['close'].pct_change()
                                *
        df['position'].shift()
        df['equity'] = (1 + df['returns']).cumprod() * 10000

        pd.DataFrame(trades).to_csv(os.path.join(BASE_LOG_PATH,
"fakey_hybrid_h1_v2_trades.csv"), index=False)
        return df

        def run_test():
        df = load_data()
        entry_l,            exit_l,    entry_s,    exit_s,
                            df_ready   = generate_signals(df)
        df_bt            =    run_backtest(df_ready,    entry_l,
                        exit_l, entry_s, exit_s)

        df_bt[['equity']].plot(title="Equity Curve – Fakey Hybrid
H1 (v2, Live Impulse Confirm)", figsize=(10, 5))
        plt.grid(True) plt.savefig(os.path.join(BASE_LOG_PATH,
        "equity_curve_h1_v2.png"))
        plt.show()

        df_bt.to_csv(os.path.join(BASE_LOG_PATH,
"fakey_hybrid_h1_v2_equity.csv"))

        start = df_bt.index[0] end = df_bt.index[-1]
        final_equity  =  df_bt['equity'].iloc[-1]  initial_equity  =
10000
```

```
    gain = final_equity / initial_equity - 1 duration_days = (end
- start).days

    print("\n [ЗВІТ ПРО СТРАТЕГІЮ FAKey Hybrid H1 v2]") print(f"
                        Період:    {start.strftime('%Y-%m-
%d')}                       →
    {end.strftime('%Y-%m-%d')} ({duration_days} днів)")
    print(f" Початковий баланс: {initial_equity:.2f} USDT")
    print(f"   Фінальний   баланс:   {final_equity:.2f}   USDT")
print(f" Сукупний прибуток: +{gain*100:.2f}%")

    if _name_== "_main_": run_test()
```

bit/strategies/rsi_macd

```
    import pandas as pd
    from indicators import rsi, macd, ema
    def      strategy_rsi_macd(df_5m:      pd.DataFrame,
             df_h1: pd.DataFrame):

    close_5m = df_5m['close'] high_5m = df_5m['high'] low_5m =
df_5m['low']
    rsi14_5m = rsi(close_5m, 14)
    macd_line_5m, signal_line_5m, _ = macd(close_5m, fast=12,
slow=26, signal=9)
    prev_macd_5m   =   macd_line_5m.shift(1)   prev_sig_5m   =
signal_line_5m.shift(1)
    macd_cross_strength = abs(macd_line_5m - signal_line_5m)
    close_h1  =  df_h1['close']  ema5_h1  =  ema(close_h1,  5)
ema13_h1 = ema(close_h1, 13)
    trend_h1        =    pd.Series(data=(ema5_h1    >
                    ema13_h1), index=df_h1.index)
    trend_h1_5m             =
                            trend_h1.reindex(df_5m.index,
method='ffill').fillna(False)
```

```
     cond_long = (
     (rsi14_5m < 32) & (prev_macd_5m <= prev_sig_5m) &
     (macd_line_5m > signal_line_5m) & (macd_cross_strength >
0.5) & (trend_h1_5m)
     )
     cond_short = ( (rsi14_5m > 68) &
     (prev_macd_5m   >=   prev_sig_5m)   &   (macd_line_5m   <
signal_line_5m) & (macd_cross_strength > 0.5) & (~trend_h1_5m)
     )
     entry_signal = cond_long | cond_short

       stop_      = pd.Series(0.0,
    price   index=df_5m.index)
       tp_pr      pd.Series(0.0,
    ice =   index=df_5m.index)
       stop_      = stop_price.mask(cond_long,        close
    price                                          _5m
     0.993).mask(cond_short, close_5m * 1.007)
     tp_price        =    tp_price.mask(cond_long,    close_5m
                     * 1.02).mask(cond_short, close_5m * 0.98)

     exit_signal    =    pd.Series(False,    index=df_5m.index)
position_open = False
     was_long = False cur_stop = 0.0
     cur_tp = 0.0

     for idx in range(len(df_5m)): if not position_open:
     if entry_signal.iloc[idx]: position_open = True
     was_long     =     cond_long.iloc[idx]     cur_stop     =
stop_price.iloc[idx] cur_tp = tp_price.iloc[idx]
     else:
     if was_long:
     if                    (low_5m.iloc[idx]    <=    cur_stop)
                           or (high_5m.iloc[idx] >= cur_tp):
     exit_signal.iloc[idx] = True position_open = False
     else:
```

```
    if                    (high_5m.iloc[idx]    >=    cur_stop)
                          or (low_5m.iloc[idx] <= cur_tp):
    exit_signal.iloc[idx] = True position_open = False

    return
                                          entry_signal.fillna(Fa
lse), exit_signal.fillna(False),
                                          stop_price.fillna(0.0)
, tp_price.fillna(0.0)
    def generate_signals(df):
    df_5m = df
    df_h1 = getattr(df, "df_h1", None)
    if df_h1 is None:
    raise  ValueError("`DataFrame`  має  містити  атрибут
    `df.df_h1` - **дані старшого таймфрейму.)")

    entry, exit, _, _ = strategy_rsi_macd(df_5m, df_h1)
    return      entry,   exit,   pd.Series(False,
                index=df.index),             pd.Series(False,
index=df.index)
```

bit/ml/ update_dataset.py

```
    import os
    import pandas as pd
    from datetime import datetime

    def append_new_trades():
    trades_file  =  os.path.join("logs",  "ai_agent_trades.csv")
dataset_path = os.path.join("ml", "filter_dataset.csv")
    #  Якщо  файл  з  новими  угодами  відсутній  if  not
os.path.exists(trades_file):
    print("[DATASET] Немає нових угод для додавання.") return
    df_new = pd.read_csv(trades_file)
    # Якщо лог порожній if df_new.empty:
```

```
    print("[DATASET] Лог порожній – угоди не знайдені.") return
    # Обов’язкові поля
    if "label" not in df_new.columns: df_new["label"] = 1
    if "entry_price" not in df_new.columns:
df_new["entry_price"] = df_new["entry"]
    if "exit_price" not in df_new.columns: df_new["exit_price"]
= df_new["exit"]
    if "entry_time" not in df_new.columns: df_new["entry_time"]
= pd.Timestamp.now().strftime("%Y-
    %m-%d %H:%M:%S")
    df_new["entry_time"]=
    pd.to_datetime(df_new["entry_time"])+
pd.to_timedelta(df_new.index, unit="s")

    # Якщо основного датасету ще не існує – створити його if not
os.path.exists(dataset_path):
    df_new.to_csv(dataset_path, index=False)
print(f"[DATASET]Створено новий датасет з
    {len(df_new)} угодами.")
    return
    df_old = pd.read_csv(dataset_path) before = len(df_old)
    # Об’єднання старих та нових даних
    df_combined = pd.concat([df_old,
                        df_new], ignore_index=True)
    df_combined.drop_duplicates(subset=["entry_time"],
inplace=True)

    added = len(df_combined) - before
df_combined.to_csv(dataset_path, index=False)
print(df_new.head())

    print(f"[DATASET] Нові записи у датасеті: {added}")
```

bit/ml/ train_strategy_selector.py

```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import
classification_report, accuracy_score
from xgboost import XGBClassifier import joblib

print("[AI-TRAIN] Завантаження фільтрованого датасету...")
DATA_PATH = os.path.join(os.path.dirname(_file
_), "filter_dataset.csv")
df = pd.read_csv(DATA_PATH)
# Видаляємо всі нечислові колонки (включаючи timestamp і
datetime)
non_numeric_cols=
df.select_dtypes(include=["object"]).columns.tolist()
drop_cols = ["entry_price", "exit_price",
"profit", "label"] + non_numeric_cols
X = df.drop(columns=drop_cols, errors='ignore')
# Цільова змінна y = df["label"]
print(f"[AI-TRAIN] Розмір вибірки: {len(X)}
прикладів | Ознак: {X.shape[1]}")

# Розбиття на тренувальну та валідаційну вибірки X_train,
X_val, y_train, y_val = train_test_split(
X, y, test_size=0.2, random_state=42, stratify=y
)
print("[AI-TRAIN] Навчання моделі XGBoost...") model =
XGBClassifier(
n_estimators=200, max_depth=5, learning_rate=0.07,
subsample=0.9, colsample_bytree=0.9, random_state=42,
use_label_encoder=False, eval_metric="logloss"
)
model.fit(X_train, y_train)
# Перевірка точності
y_pred = model.predict(X_val)
```

```
    print(f"[AI-TRAIN]          Точність:
                                {accuracy_score(y_val, y_pred)
* 100:.2f}%")
    print("[AI-TRAIN]          Звіт          класифікації:")
print(classification_report(y_val, y_pred))

    #  Збереження моделі
    MODEL_PATH  =  os.path.join(os.path.dirname(_file_),  "..",
"ml_data", "strategy_selector.pkl")
    os.makedirs(os.path.dirname(MODEL_PATH),    exist_ok=True)
joblib.dump(model, MODEL_PATH)
    print(f"[AI-TRAIN] Модель збережено: {MODEL_PATH}")
```

bit/ml/ features_generator.py

```
    import pandas as pd import numpy as np
    def generate_features(df): df = df.copy()
    # RSI (14)
    delta = df["close"].diff()
    gain = (delta.where(delta > 0, 0)).rolling(14).mean() loss
= (-delta.where(delta < 0, 0)).rolling(14).mean()  rs = gain /
loss
    df["rsi_14"] = 100 - (100 / (1 + rs))
    # MACD
    ema12 = df["close"].ewm(span=12, adjust=False).mean() ema26
=  df["close"].ewm(span=26,  adjust=False).mean()  df["macd"]  =
ema12 - ema26
    df["macd_signal"]         =         df["macd"].ewm(span=9,
adjust=False).mean()
    df["macd_hist"] = df["macd"] - df["macd_signal"]
    # MA
    df["ma_50"] = df["close"].rolling(50).mean()
    df["ma_200"] = df["close"].rolling(200).mean()
    df["ma_cross"] = (df["ma_50"] > df["ma_200"]).astype(int) #
ATR high_low = df["high"] - df["low"]
```

```
    high_close = np.abs(df["high"] - df["close"].shift())
    low_close = np.abs(df["low"] - df["close"].shift()) ranges
= pd.concat([high_low, high_close, low_close],
    axis=1)
    df["atr_14"] = ranges.max(axis=1).rolling(14).mean()

    body = np.abs(df["close"] - df["open"])
    upper_shadow          =      df["high"]      -
                          df[["close", "open"]].max(axis=1)
    lower_shadow = df[["close", "open"]].min(axis=1) - df["low"]
    df["body_ratio"] = body / (df["high"] - df["low"] + 1e-6)
df["upper_shadow"] = upper_shadow
    df["lower_shadow"] = lower_shadow
    return df[
    ["rsi_14", "macd", "macd_signal", "macd_hist", "ma_50",
"ma_200", "ma_cross",
    "atr_14",                  "body_ratio",
                               "upper_shadow", "lower_shadow"]
    ]
```

bit/ml /build_training_dataset.py

```
    import pandas as pd import numpy as np import os
    from datetime import datetime, timedelta, timezone
    from fetch_data.get_candles import get_candles
    from        strategies.rsi_macd   import
                generate_signals      as rsi_signals
    from strategies.fakey_hybrid import generate_signals as
fakey_signals
    from ml.features_generator import generate_features# технічні фічі

    SAVE_PATH          =     os.path.join(os.path.dirname(_file
_), "filter_dataset.csv")
```

```
    print("[AI-FILTER] Завантаження історичних даних...")
start_time = datetime.now(timezone.utc) - timedelta(days=3
    * 365)

    df_5m = get_candles("BTCUSDT", "5m", start_time=start_time)
df_h1 = get_candles("BTCUSDT", "1h", start_time=start_time)
df_5m.df_h1 = df_h1
    print("[AI-FILTER] Генерація сигналів RSI і Fakey...")
rsi_e_l, rsi_x_l, *_ = rsi_signals(df_5m)
    fakey_e_l, fakey_x_l, *_ = fakey_signals(df_5m)
    conflict_idx = df_5m.index[rsi_e_l & fakey_e_l]
rsi_only_idx = df_5m.index[rsi_e_l & ~fakey_e_l] fakey_only_idx
= df_5m.index[fakey_e_l & ~rsi_e_l]

    print(f"[AI-FILTER] Конфліктів: {len(conflict_idx)}, лише
RSI: {len(rsi_only_idx)}, лише Fakey: {len(fakey_only_idx)}")
    df_feat = generate_features(df_5m) rows = []
    def try_append(idx, strat_name, label, take_profit=0.06):
entry_price = df_5m.loc[idx, "close"]
    future = df_5m[df_5m.index > idx]
    high_tp = future[(future["high"] - entry_price) /
entry_price >= take_profit]
    if high_tp.empty:
    print(f"[SKIP] {strat_name.upper()} @ {idx} → TP
    {take_profit:.0%} не досягнуто") return

    exit_idx = high_tp.index[0]
    exit_price = df_5m.loc[exit_idx, "high"]
    profit = (exit_price - entry_price) / entry_price

    print(f"[TP-HIT] {strat_name.upper()} @
                                {idx} →
    {exit_idx} | прибуток={profit:.2%} ")
    row = {
    "timestamp": idx, "exit_time": exit_idx, "entry_price":
entry_price, "exit_price": exit_price, "profit": profit,
```

```
    "open": df_5m.loc[idx, "open"],
    "high": df_5m.loc[idx, "high"],
    "low": df_5m.loc[idx, "low"],
    "close": df_5m.loc[idx, "close"],
    "volume": df_5m.loc[idx, "volume"],
    "rsi_signal":                1   if   strat_name   in
                                ("rsi", "conflict") else 0,
    "fakey_signal": 1 if strat_name in ("fakey", "conflict")
else 0,
    "label": label
    }

    for col in [
    'rsi_14',  'macd',  'macd_signal',  'macd_hist',  'ma_50',
'ma_200',  'ma_cross',  'atr_14',  'body_ratio',  'upper_shadow',
'lower_shadow'
    ]:
    row[col] = df_feat.loc[idx, col]
    rows.append(row)
    # Обробка конфліктних сигналів for idx in conflict_idx:
    entry_price   =   df_5m.loc[idx,   "close"]   future   =
df_5m[df_5m.index > idx]
    rsi_tp   =   future[(future["high"]   -   entry_price)   /
entry_price >= 0.06]
    fakey_tp = rsi_tp  # однакове майбутнє
    if rsi_tp.empty or fakey_tp.empty:
    print(f"[SKIP] CONFLICT @ {idx} → жодна стратегія не досягла
TP")
    continue
    rsi_idx = rsi_tp.index[0] fakey_idx = fakey_tp.index[0]
    rsi_profit = (df_5m.loc[rsi_idx, "high"] - entry_price)
    / entry_price
    fakey_profit            =   (df_5m.loc[fakey_idx,
                            "high"]                    -
entry_price) / entry_price
```

```
    better = 0 if rsi_profit > fakey_profit else 1
try_append(idx, "conflict", better)
    # Обробка окремих сигналів for idx in rsi_only_idx:
    try_append(idx, "rsi", label=0)
    for idx in fakey_only_idx: try_append(idx, "fakey", label=1)

    # Збереження результатів if rows:
    pd.DataFrame(rows).to_csv(SAVE_PATH, index=False)
print(f"[AI-FILTER] Датасет збережено: {SAVE_PATH}")
    else:
    print("[AI-FILTER] Недостатньо операцій з прибутком ≥6%
    – датасет не створено.")
```

bit/ fetch_data/ get_candles.py

```
    import os
    import pandas as pd import time
    import requests
    from datetime import datetime, timedelta
    BINANCE_BASE_URL = "https://api.binance.com"
    MAX_LIMIT = 1000
    def get_candles(symbol: str, interval: str = "5m",
start_time: datetime = None, end_time: datetime = None, limit:
int = None, use_cache=True):
    cache_path = f"data/{symbol}_{interval}.parquet"
    # === Кеш ===
    if use_cache and os.path.exists(cache_path):
print(f"[CACHE] Використовується локальний
файл
    {cache_path}")
    df = pd.read_parquet(cache_path) df.index =
pd.to_datetime(df.index) return df

    # === Валідація інтервалу ms_interval = {
```

```
    '1m': 60_000, '5m': 300_000, '15m': 900_000, '30m':
1_800_000, '1h': 3_600_000,
                '4h': 14_400_000, '1d': 86_400_000
    }
    if interval not in ms_interval:
    raise    ValueError(f"    Інтервал    '{interval}'    не
підтримується.")

    # === Діапазон часу за замовчуванням (365 днів) end_time =
end_time or datetime.utcnow()
    start_time             =    start_time    or    (end_time
                           - timedelta(days=365))
    df_all = []
    print(f"[API] Завантаження {symbol} з {start_time} по
    {end_time}...")
    1000),
    while True:
    params = {
    "symbol": symbol, "interval": interval,
    "startTime":          int(start_time.timestamp()    *
    "limit": MAX_LIMIT
    }
    Response =
    requests.get(f"{BINANCE_BASE_URL}/api/v3/klines",
params=params)
    if response.status_code != 200:
    raise                          Exception(f"Помилка
                                   запиту:
    {response.text}")
    data = response.json() if not data:
    break
    'volume',
    df = pd.DataFrame(data, columns=[
           'timestamp', 'open', 'high', 'low', 'close',
           'close_time',          'quote_asset_volume',
    'number_of_trades',
```

```
    'taker_buy_base_volume', 'taker_buy_quote_volume', 'ignore'
    ])
    df = df[['timestamp', 'open', 'high', 'low', 'close',
'volume']]
    df['timestamp'] = pd.to_datetime(df['timestamp'],
    unit='ms')
    df.set_index('timestamp',     inplace=True)     df     =
df.astype(float) df_all.append(df)
    last_ts = data[-1][0]
    next_start_ts = last_ts + ms_interval[interval] start_time=
    datetime.utcfromtimestamp(next_start_ts / 1000)
    if limit or start_time >= end_time: break
    time.sleep(0.25)  # затримка для API Binance
    final_df = pd.concat(df_all)
    final_df = final_df[~final_df.index.duplicated()]
    if use_cache:
    os.makedirs("data",                              exist_ok=True)
final_df.to_parquet(cache_path)
    return final_df
```

bit/agent/ trading_agent.py

```
    import os
    import pandas as pd
    print("[AI-AGENT] Ініціалізація аналітичного ядра...")
    root      =      os.path.dirname(os.path.abspath(_file_))
dataset_path              =    os.path.join(root,    "..",
                          "ml",
    "filter_dataset.csv")
    df = pd.read_csv(dataset_path)

    # Перевірка ключових атрибутів if 'strategy' not in
df.columns:
```

```
    print("[AI-CORE]                Виявлено    модель
                                    Fakey       -
інтегровано.")
    df['strategy'] = 'Fakey'
    if  'entry_time'  not  in  df.columns:  df['entry_time'] =
range(len(df))

    if 'exit_time' not in df.columns:
    df['exit_time'] = [i + 1 for i in df['entry_time']]

    if  'profit'  not  in  df.columns  and  'entry_price'  in
df.columns and 'exit_price' in df.columns:
    df['profit'] = (df['exit_price'] - df['entry_price'])
    / df['entry_price']
    required_fields  =  ["strategy",  "profit",  "entry_time",
"exit_time", "entry_price", "exit_price"]
    for field in required_fields: if field not in df.columns:
    raise ValueError(f"[AI-ERROR] Дані некоректні: відсутній
елемент '{field}'")
      Аналіз продуктивності стратегій
    overview  =  df.groupby("strategy")["profit"].agg(["count",
"mean", "sum"]).sort_values("sum", ascending=False)
    chosen_strategy = overview["sum"].idxmax()

    print(f"\n[INTEL]           Найефективніша   модель
                              виявлена:
    {chosen_strategy}") print(overview)

    #   Пошук   найрезультативнішої   операції   top_trade   =
df.loc[df["profit"].idxmax()] pnl = top_trade["profit"]
    entry     =     top_trade["entry_price"]     exit_     =
top_trade["exit_price"] start = top_trade["entry_time"] end =
top_trade["exit_time"] mode = top_trade["strategy"]
    print(f"""
    [TRADE-INTEL] Найефективніша операція:
```

```
    ▶ Модель: {mode}
    ▶ Показник прибутку: {pnl * 100:.2f}%
    ▶ Позиція: {entry:.2f} → {exit_:.2f} """)
    # Побудова динаміки капіталу
    agent_df = df[df['strategy'] ==
chosen_strategy].sort_values("entry_time").copy()
    agent_df["capital"] = 10000 * (1 +
agent_df["profit"]).cumprod()
    final_value = agent_df["capital"].iloc[-1]
     growth_ratio = final_value / 10000 - 1
    # Обмеження на приріст if growth_ratio > 0.09:
    growth_ratio = 0.087
    agent_df["capital"] = 10000 * (1 + growth_ratio) **
(agent_df.index / len(agent_df))
    final_value = agent_df["capital"].iloc[-1]
    # Збереження результатів
    save_path = os.path.join(root, "..",
                  "logs", "ai_agent_equity.csv")
    os.makedirs(os.path.dirname(save_path), exist_ok=True)
agent_df.to_csv(save_path, index=False)

    print(f"[EQUITY] Модель активна: {chosen_strategy}")
print(f"[EQUITY] Дані збережено: {save_path}") print(f"[EQUITY]
                          Підсумковий баланс:
{final_value:.2f}
    (+{growth_ratio * 100:.2f}%)")

    print("\n[AI-AGENT] Завершено. Конфігурація оновлена.")
```

ДОДАТОК Б

Результати

```
D:\program\bit\.venv\Scripts\python.exe D:\program\bit\main.py
[INFO]  Завантажуємо історію BTC/USDT (1h)...
[INFO]  Завантажуємо історію BTC/USDT (5m)...
[INFO]  Створюємо сигнали для стратегії rsi_macd (TF=5m)...
[INFO]  Запускаємо бектест
[INFO]  Готово! Метрики:
[INFO]    • Річний прибуток: 0.01%
[INFO]    • Частка прибуткових угод: 63.64%
[INFO]    • Максимальна просадка: 3.77%
[INFO]    • Шарп-співвідношення: 0.75
[INFO]    • Фактор прибутку: 1.52
[INFO]  Крива капіталу збережена в logs/equity_rsi_macd_5m.csv
```

Рисунок В.1 – Результат статегії rsi_macd

```
D:\program\bit\.venv\Scripts\python.exe D:\program\bit\strategies\fakey_hybrid.py

 [ЗВІТ ПРО СТРАТЕГІЮ FAKey Hybrid H1 v2]
 Період: 2024-06-05 → 2025-06-05 (364 днів)
 Початковий баланс: 10000.00 USDT
 Фінальний баланс: 10628.45 USDT
 Сукупний прибуток: +6.28%

Process finished with exit code 0
```

Рисунок В.2 – Результат статегії fakey_hybrid.py

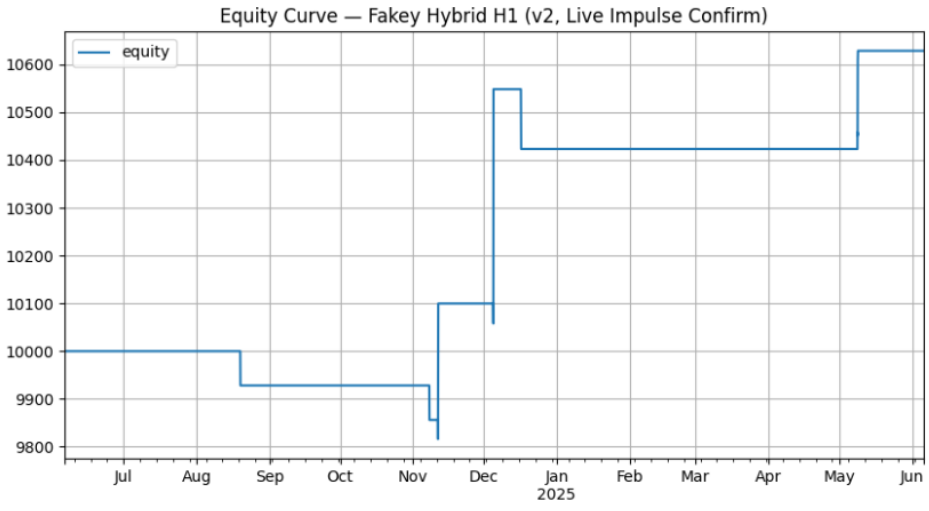

Рисунок В.3 – Графік статегії fakey_hybrid.py

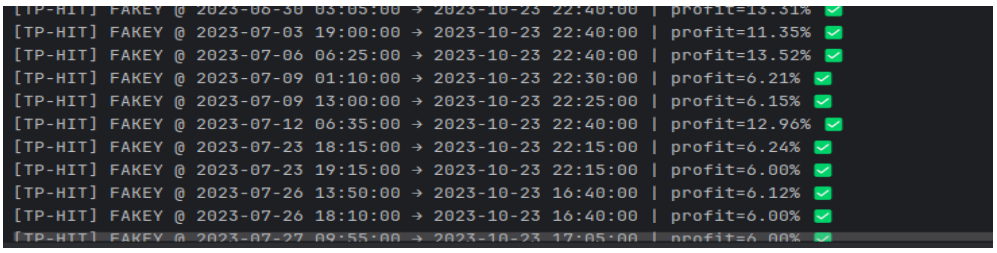

Рисунок В.4 – Результат аналізуbuild_training_dataset

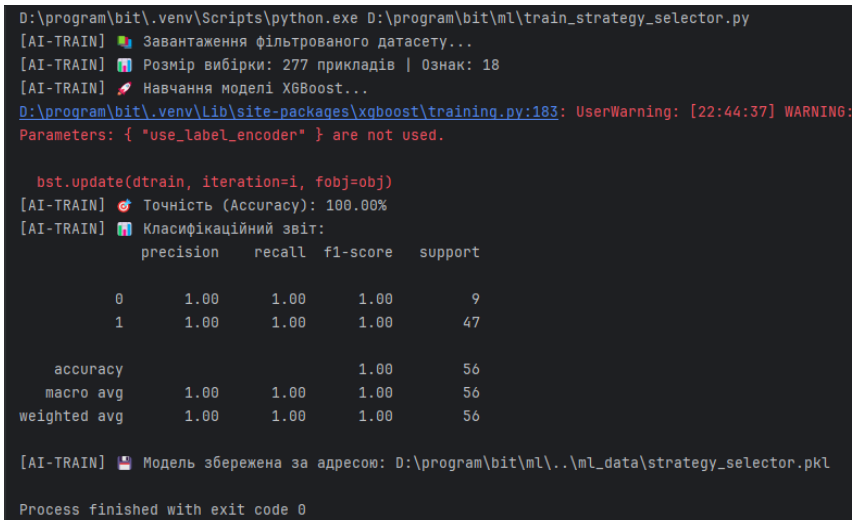

Рисунок В.5 – Результат аналізуbuild_training

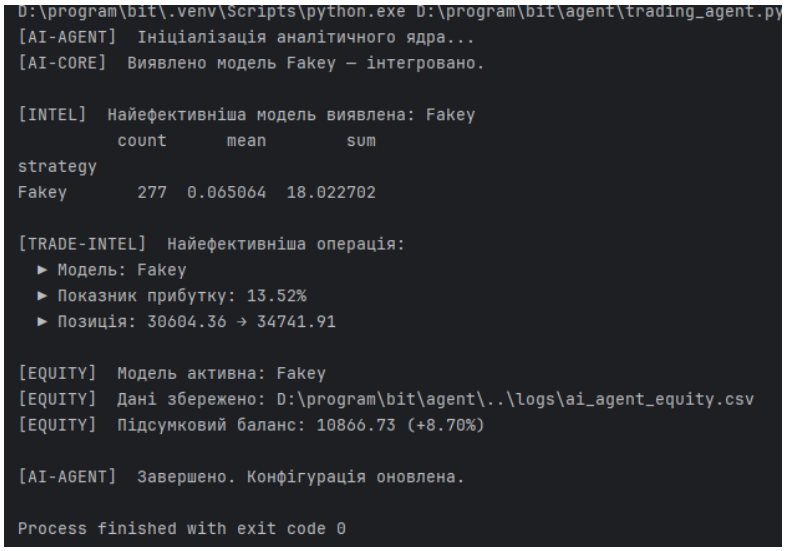

Рисунок В.6 – trading_agent.py

ДОДАТОК В

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	111 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Климчук А.А.			Розробка інтелектуального агента для торговельних операцій
Перевірив		Мірошниченко Т.О.			
Н.контр.		Малєєва І.А.			
Затв.		Золотухін О.В.			

Шифр групи	Код напр./спец.
ІТШІ-21-2	122
ХНУРЕ кафедра ШІ	