

```

import json
import hmac
import hashlib
import time
import threading
import urllib.request
import urllib.parse
import numpy as np
import websocket
import logging
import requests
import os
import math
import traceback
from datetime import datetime
from concurrent.futures import ThreadPoolExecutor

# ===== CẤU HÌNH LOGGING =====
def setup_logging():
    logging.basicConfig(
        level=logging.INFO,
        format='%(asctime)s - %(levelname)s - %(module)s - %(message)s',
        handlers=[
            logging.StreamHandler(),
            logging.FileHandler('bot_errors.log')
        ]
    )
    return logging.getLogger()

logger = setup_logging()

# ===== HÀM TELEGRAM =====
def send_telegram(message, chat_id=None, reply_markup=None, bot_token=None, default_chat_id=None):
    if not bot_token:
        logger.warning("Telegram Bot Token chưa được thiết lập")
        return

    chat_id = chat_id or default_chat_id
    if not chat_id:
        logger.warning("Telegram Chat ID chưa được thiết lập")
        return

    url = f"https://api.telegram.org/bot{bot_token}/sendMessage"
    payload = {
        "chat_id": chat_id,
        "text": message,

```

```

        "parse_mode": "HTML"
    }

    if reply_markup:
        payload["reply_markup"] = json.dumps(reply_markup)

    try:
        response = requests.post(url, json=payload, timeout=15)
        if response.status_code != 200:
            logger.error(f"Lỗi Telegram ({response.status_code}): {response.text}")
    except Exception as e:
        logger.error(f"Lỗi kết nối Telegram: {str(e)}")

# ===== SMART EXIT MANAGER =====
class SmartExitManager:
    """QUẢN LÝ THÔNG MINH 4 CƠ CHẾ ĐÓNG LỆNH"""

    def __init__(self, bot_instance):
        self.bot = bot_instance
        self.config = {
            'enable_trailing': False,
            'enable_time_exit': False,
            'enable_volume_exit': False,
            'enable_support_resistance': False,
            'trailing_activation': 30,
            'trailing_distance': 15,
            'max_hold_time': 6,
            'min_profit_for_exit': 10
        }

        self.trailing_active = False
        self.peak_price = 0
        self.position_open_time = 0
        self.volume_history = []

        # Thêm các cấu hình nâng cấp từ v2_part_2 (được tích hợp trong
        _check_trailing_stop)
        self.config.update({
            'breakeven_at': 12, # Được dùng trong BaseBot check_tp_sl
            của v2_part2, nhưng sẽ dùng lại logic trong v16.py
            'trail_adaptive': True, # Cần chỉ báo ATR để dùng
            'tp_ladder': [ # Cần logic partial_close
                {'roi': 15, 'pct': 0.30},
                {'roi': 25, 'pct': 0.30},
                {'roi': 40, 'pct': 0.40},
            ]
        })

```

```

self._breakeven_active = False
self._tp_hit = set()

def update_config(self, **kwargs):
    """Cập nhật cấu hình từ người dùng"""
    changed = {}
    for key, value in kwargs.items():
        if key in self.config:
            self.config[key] = value
            changed[key] = value
    if changed:
        self.bot.log(f"⚙️ Cập nhật Smart Exit: {changed}")

def _calculate_roi(self, current_price):
    """Tính ROI hiện tại"""
    if not self.bot.position_open or self.bot.entry <= 0 or
abs(self.bot.qty) <= 0:
        return 0.0
    if self.bot.side == "BUY":
        # Tính PnL dựa trên giá hiện tại và giá vào
        profit = (current_price - self.bot.entry) *
abs(self.bot.qty)
    else:
        profit = (self.bot.entry - current_price) *
abs(self.bot.qty)

    # Tính vốn đầu tư thực tế (dùng cho ROI)
    invested = self.bot.entry * abs(self.bot.qty) / self.bot.lev
    if invested <= 0: return 0.0

    return (profit / invested) * 100.0

def check_all_exit_conditions(self, current_price,
current_volume=None):
    """KIỂM TRA TẤT CẢ ĐIỀU KIỆN ĐÓNG LỆNH"""
    if not self.bot.position_open:
        return None

    exit_reasons = []
    current_roi = self._calculate_roi(current_price)

    # 1. BREAKEVEEN + TP LADDER (Logic nâng cấp từ v2_part2)
    if (not self._breakeven_active) and (current_roi >=
self.config.get('breakeven_at', 12)):
        self._breakeven_active = True
        # Đảm bảo min_profit_for_exit không giảm dưới 0 khi
breakeven

```

```

        self.config['min_profit_for_exit'] =
max(self.config.get('min_profit_for_exit', 10), 0)
        self.bot.log(f"██ Kịch hoạt Breakeven tại ROI
{current_roi:.1f}%")

        # Logic TP Ladder (cần hàm partial_close được tích hợp vào
BaseBot)
        # Bỏ qua vì partial_close không có trong phiên bản v16.py gốc
mà BaseBot kế thừa
        # for step in self.config.get('tp_ladder', []):
        #     roi_lv = step.get('roi', 0); pct = step.get('pct', 0)
        #     key = f"tp_{roi_lv}"
        #     if current_roi >= roi_lv and key not in self._tp_hit:
        #         # Thao tác partial_close bị thiếu trong BaseBot của
v16.py
        #         # ok = self.bot.partial_close(pct, reason=f"TP
ladder {roi_lv}%")
        #         # if ok:
        #         #     self._tp_hit.add(key)
        #         pass

        # 2. TRAILING STOP EXIT
        if self.config['enable_trailing']:
            reason = self._check_trailing_stop(current_price)
            if reason:
                exit_reasons.append(reason)

        # 3. TIME-BASED EXIT
        if self.config['enable_time_exit']:
            reason = self._check_time_exit()
            if reason:
                exit_reasons.append(reason)

        # 4. VOLUME-BASED EXIT
        if self.config['enable_volume_exit'] and current_volume:
            reason = self._check_volume_exit(current_volume)
            if reason:
                exit_reasons.append(reason)

        # 5. SUPPORT/RESISTANCE EXIT
        if self.config['enable_support_resistance']:
            reason = self._check_support_resistance(current_price)
            if reason:
                exit_reasons.append(reason)

        # Chỉ đóng lệnh nếu đang có lãi đạt ngưỡng tối thiểu HOẶC
Breakeven đã kích hoạt
        # Nếu Breakeven active, ngưỡng lãi tối thiểu có thể là 0

```

```

        min_profit = self.config['min_profit_for_exit'] if not
self._breakeven_active else 0

        if exit_reasons:
            if current_roi >= min_profit:
                return f"Smart Exit: {' + '.join(exit_reasons)} | Lãi:
{current_roi:.1f}%"

        return None

def _check_trailing_stop(self, current_price):
    """Trailing Stop - Bảo vệ lợi nhuận"""
    current_roi = self._calculate_roi(current_price)
    distance = self.config['trailing_distance']

    # Kích hoạt trailing khi đạt ngưỡng
    if current_roi >= self.config['trailing_activation'] and not
self.trailing_active:
        self.trailing_active = True
        self.peak_price = current_price
        self.bot.log(f"🟢 Kích hoạt Trailing Stop | Lãi
{current_roi:.1f}%",

    # Cập nhật đỉnh mới
    if self.trailing_active:
        # Logic adaptive trailing bị thiếu do cần chỉ báo ATR từ
v2_part1
        # if self.config.get('trail_adaptive'):
        #     ...

        # Cập nhật peak price
        if self.bot.side == "BUY":
            self.peak_price = max(self.peak_price, current_price)
            trigger_price = self.peak_price * (1 - distance /
100.0)

            if current_price <= trigger_price:
                return f"▼ Trailing hit ({distance:.1f}%"
            else: # SELL side
                self.peak_price = min(self.peak_price, current_price)
                trigger_price = self.peak_price * (1 + distance /
100.0)

                if current_price >= trigger_price:
                    return f"▼ Trailing hit ({distance:.1f}%"

        return None

def _check_time_exit(self):
    """Time-based Exit - Giới hạn thời gian giữ lệnh"""

```

```

if self.position_open_time == 0:
    return None

holding_hours = (time.time() - self.position_open_time) / 3600

if holding_hours >= self.config['max_hold_time']:
    return f"Time({holding_hours:.1f}h)"

return None

def _check_volume_exit(self, current_volume):
    """Volume-based Exit - Theo dấu hiệu volume"""
    if len(self.volume_history) < 5:
        self.volume_history.append(current_volume)
        return None

    # Logic đơn giản từ v16.py:
    avg_volume = sum(self.volume_history[-5:]) / 5

    if current_volume < avg_volume * 0.4:
        return "Volume(giảm 60%)"

    self.volume_history.append(current_volume)
    if len(self.volume_history) > 10:
        self.volume_history.pop(0)

    return None

def _check_support_resistance(self, current_price):
    """Support/Resistance Exit - Theo key levels"""
    # Logic đơn giản từ v16.py:
    if self.bot.side == "BUY":
        target_profit = 5.0
        target_price = self.bot.entry * (1 + target_profit/100)

        if current_price >= target_price:
            return f"Resistance(+{target_profit}%)"

    return None

def on_position_opened(self):
    """Khi mở position mới"""
    self.trailing_active = False
    self.peak_price = self.bot.entry
    self.position_open_time = time.time()
    self.volume_history = []
    self._breakeven_active = False
    self._tp_hit.clear()

```

```

# ===== MENU TELEGRAM HOÀN CHỈNH =====
def create_main_menu():
    return {
        "keyboard": [
            [{"text": "📊 Danh sách Bot"}],
            [{"text": "➕ Thêm Bot"}, {"text": "⊖ Dừng Bot"}],
            [{"text": "💰 Số dư"}, {"text": "📈 Vị thế"}],
            [{"text": "⚙️ Cấu hình"}, {"text": "🎯 Chiến lược"}]
        ],
        "resize_keyboard": True,
        "one_time_keyboard": False
    }

def create_cancel_keyboard():
    return {
        "keyboard": [{"text": "❌ Hủy bỏ"}],
        "resize_keyboard": True,
        "one_time_keyboard": True
    }

def create_strategy_keyboard():
    return {
        "keyboard": [
            [{"text": "🤖 RSI/EMA Recursive"}, {"text": "📊 EMA Crossover"}],
            [{"text": "🎯 Reverse 24h"}, {"text": "📈 Trend Following"}],
            [{"text": "⚡ Scalping"}, {"text": "🛡️ Safe Grid"}],
            [{"text": "🔄 Bot Động Thông Minh"}, {"text": "❌ Hủy bỏ"}]
        ],
        "resize_keyboard": True,
        "one_time_keyboard": True
    }

def create_exit_strategy_keyboard():
    """Bàn phím chọn chiến lược thoát lệnh"""
    return {
        "keyboard": [
            [{"text": "🔄 Thoát lệnh thông minh"}, {"text": "⚡ Thoát lệnh cơ bản"}],
            [{"text": "🎯 Chỉ TP/SL cố định"}, {"text": "❌ Hủy bỏ"}]
        ],
        "resize_keyboard": True,
        "one_time_keyboard": True
    }

```

```

def create_smart_exit_config_keyboard():
    """Bàn phím cấu hình Smart Exit"""
    return {
        "keyboard": [
            [{"text": "Trailing: 30/15"}, {"text": "Trailing: 50/20"}],
            [{"text": "Time Exit: 4h"}, {"text": "Time Exit: 8h"}],
            [{"text": "Kết hợp Full"}, {"text": "Cơ bản"}],
            [{"text": "❌ Hủy bỏ"}]
        ],
        "resize_keyboard": True,
        "one_time_keyboard": True
    }

def create_bot_mode_keyboard():
    """Bàn phím chọn chế độ bot"""
    return {
        "keyboard": [
            [{"text": "🤖 Bot Tĩnh - Coin cụ thể"}, {"text": "🔄 Bot Động - Tự tìm coin"}],
            [{"text": "❌ Hủy bỏ"}]
        ],
        "resize_keyboard": True,
        "one_time_keyboard": True
    }

def create_symbols_keyboard(strategy=None):
    """Bàn phím chọn coin"""
    try:
        symbols = get_all_usdt_pairs(limit=12)
        if not symbols:
            symbols = ["BTCUSDT", "ETHUSDT", "BNBUSDT", "ADAUSDT", "DOGEUSDT", "XRPUSDT", "DOTUSDT", "LINKUSDT"]
    except:
        symbols = ["BTCUSDT", "ETHUSDT", "BNBUSDT", "ADAUSDT", "DOGEUSDT", "XRPUSDT", "DOTUSDT", "LINKUSDT"]

    keyboard = []
    row = []
    for symbol in symbols:
        row.append({"text": symbol})
        if len(row) == 3:
            keyboard.append(row)
            row = []
    if row:
        keyboard.append(row)
    keyboard.append([{"text": "❌ Hủy bỏ"}])

```



```

return {
    "keyboard": keyboard,
    "resize_keyboard": True,
    "one_time_keyboard": True
}

def create_leverage_keyboard(strategy=None):
    """Bàn phím chọn đòn bẩy"""
    leverages = ["3", "5", "10", "15", "20", "25", "50", "75", "100"]

    keyboard = []
    row = []
    for lev in leverages:
        row.append({"text": f"{lev}x"})
        if len(row) == 3:
            keyboard.append(row)
            row = []
    if row:
        keyboard.append(row)
    keyboard.append([{"text": "✖ Hủy bỏ"}])

    return {
        "keyboard": keyboard,
        "resize_keyboard": True,
        "one_time_keyboard": True
    }

def create_percent_keyboard():
    """Bàn phím chọn % số dư"""
    return {
        "keyboard": [
            [{"text": "1"}, {"text": "3"}, {"text": "5"}, {"text":
"10"}],
            [{"text": "15"}, {"text": "20"}, {"text": "25"}, {"text":
"50"}],
            [{"text": "✖ Hủy bỏ"}]
        ],
        "resize_keyboard": True,
        "one_time_keyboard": True
    }

def create_tp_keyboard():
    """Bàn phím chọn Take Profit"""
    return {
        "keyboard": [
            [{"text": "50"}, {"text": "100"}, {"text": "200"}],
            [{"text": "300"}, {"text": "500"}, {"text": "1000"}],
            [{"text": "✖ Hủy bỏ"}]
        ]
    }

```

```

    ],
    "resize_keyboard": True,
    "one_time_keyboard": True
}

def create_sl_keyboard():
    """Bàn phím chọn Stop Loss"""
    return {
        "keyboard": [
            [{"text": "0"}, {"text": "50"}, {"text": "100"}],
            [{"text": "150"}, {"text": "200"}, {"text": "500"}],
            [{"text": "✗ Hủy bỏ"}]
        ],
        "resize_keyboard": True,
        "one_time_keyboard": True
    }

def create_threshold_keyboard():
    return {
        "keyboard": [
            [{"text": "30"}, {"text": "50"}, {"text": "70"}],
            [{"text": "100"}, {"text": "150"}, {"text": "200"}],
            [{"text": "✗ Hủy bỏ"}]
        ],
        "resize_keyboard": True,
        "one_time_keyboard": True
    }

def create_volatility_keyboard():
    return {
        "keyboard": [
            [{"text": "2"}, {"text": "3"}, {"text": "5"}],
            [{"text": "7"}, {"text": "10"}, {"text": "15"}],
            [{"text": "✗ Hủy bỏ"}]
        ],
        "resize_keyboard": True,
        "one_time_keyboard": True
    }

def create_grid_levels_keyboard():
    return {
        "keyboard": [
            [{"text": "3"}, {"text": "5"}, {"text": "7"}],
            [{"text": "10"}, {"text": "15"}, {"text": "20"}],
            [{"text": "✗ Hủy bỏ"}]
        ],
        "resize_keyboard": True,
        "one_time_keyboard": True
    }

```

```

    }

# ===== QUẢN LÝ COIN CHUNG =====
class CoinManager:
    _instance = None
    _lock = threading.Lock()

    def __new__(cls):
        with cls._lock:
            if cls._instance is None:
                cls._instance = super(CoinManager, cls).__new__(cls)
                cls._instance.managed_coins = {}
                cls._instance.position_coins = set()
                # Thêm cooldown logic từ v2_part1
                cls._instance.symbol_cooldowns = {}
                cls._instance.cooldown_seconds = 20*60 # 20 phút
            return cls._instance

    def register_coin(self, symbol, bot_id, strategy,
config_key=None):
        with self._lock:
            # Kiểm tra cooldown trước khi đăng ký
            if self.is_in_cooldown(symbol):
                return False

            if symbol not in self.managed_coins:
                self.managed_coins[symbol] = {
                    "strategy": strategy,
                    "bot_id": bot_id,
                    "config_key": config_key
                }
            return True
        return False

    def unregister_coin(self, symbol):
        with self._lock:
            if symbol in self.managed_coins:
                del self.managed_coins[symbol]
                return True
            return False

    def is_coin_managed(self, symbol):
        with self._lock:
            return symbol in self.managed_coins

    def has_same_config_bot(self, symbol, config_key):
        with self._lock:
            if symbol in self.managed_coins:

```

```

        existing_config =
self.managed_coins[symbol].get("config_key")
        return existing_config == config_key
    return False

def count_bots_by_config(self, config_key):
    with self._lock:
        count = 0
        for coin_info in self.managed_coins.values():
            if coin_info.get("config_key") == config_key:
                count += 1
        return count

def get_managed_coins(self):
    with self._lock:
        return self.managed_coins.copy()

# ===== COOLDOWN API (tích hợp từ v2_part1) =====
def set_cooldown(self, symbol, seconds=None):
    ts = time.time() + (seconds or self.cooldown_seconds)
    with self._lock:
        self.symbol_cooldowns[symbol.upper()] = ts

def is_in_cooldown(self, symbol):
    with self._lock:
        ts = self.symbol_cooldowns.get(symbol.upper())
        if not ts:
            return False
        if time.time() >= ts:
            del self.symbol_cooldowns[symbol.upper()]
            return False
        return True

def cooldown_left(self, symbol):
    with self._lock:
        ts = self.symbol_cooldowns.get(symbol.upper())
        return max(0, int(ts - time.time())) if ts else 0

# ===== API BINANCE =====
BASE_FAPI = "https://fapi.binance.com" # Thêm từ v2_part1

def sign(query, api_secret):
    try:
        return hmac.new(api_secret.encode(), query.encode(),
hashlib.sha256).hexdigest()
    except Exception as e:
        logger.error(f"Lỗi tạo chữ ký: {str(e)}")
        return ""

```

```

def signed_request(url_path, params, api_key, secret_key,
method='GET'):
    # Hàm từ v2_part1, được dùng trong v2_part2
    query = urllib.parse.urlencode(params)
    signature = hmac.new(secret_key.encode('utf-8'),
query.encode('utf-8'), hashlib.sha256).hexdigest()
    headers = {"X-MBX-APIKEY": api_key}
    url = f"{BASE_FAPI}{url_path}?{query}&signature={signature}"
    req = urllib.request.Request(url, headers=headers, method=method)
    with urllib.request.urlopen(req, timeout=10) as resp:
        return json.loads(resp.read().decode())

def binance_api_request(url, method='GET', params=None, headers=None):
    # Hàm gốc từ v16.py, giữ lại vì nó có logic retry/error handling
    tốt hơn
    max_retries = 3
    for attempt in range(max_retries):
        try:
            if method.upper() == 'GET':
                if params:
                    query = urllib.parse.urlencode(params)
                    url = f"{url}?{query}"
                    req = urllib.request.Request(url, headers=headers or
{ })
                else:
                    data = urllib.parse.urlencode(params).encode() if
params else None
                    req = urllib.request.Request(url, data=data,
headers=headers or { }, method=method)

            with urllib.request.urlopen(req, timeout=15) as response:
                if response.status == 200:
                    return json.loads(response.read().decode())
                else:
                    error_content = response.read().decode()
                    logger.error(f"Lỗi API ({response.status}):
{error_content}")

                    if response.status == 401:
                        return None
                    if response.status == 429:
                        time.sleep(2 ** attempt)
                    elif response.status >= 500:
                        time.sleep(1)
                    continue
        except urllib.error.HTTPError as e:
            logger.error(f"Lỗi HTTP ({e.code}): {e.reason}")
            if e.code == 401:

```

```

        return None
    if e.code == 429:
        time.sleep(2 ** attempt)
    elif e.code >= 500:
        time.sleep(1)
    continue
except Exception as e:
    logger.error(f"Lỗi kết nối API: {str(e)}")
    time.sleep(1)

logger.error(f"Không thể thực hiện yêu cầu API sau {max_retries}
lần thử")
return None

def get_all_usdt_pairs(limit=100):
    # Dùng hàm gốc từ v16.py/v2_part1
    try:
        url = f"{BASE_FAPI}/fapi/v1/exchangeInfo"
        data = binance_api_request(url)
        if not data:
            logger.warning("Không lấy được dữ liệu từ Binance, trả về
danh sách rỗng")
            return []

        usdt_pairs = []
        for symbol_info in data.get('symbols', []):
            symbol = symbol_info.get('symbol', '')
            # Lấy các cặp USDT Futures TRADING
            if symbol.endswith('USDT') and symbol_info.get('status')
== 'TRADING':
                usdt_pairs.append(symbol)

        logger.info(f"✅ Lấy được {len(usdt_pairs)} coin USDT từ
Binance")
        return usdt_pairs[:limit] if limit else usdt_pairs

    except Exception as e:
        logger.error(f"❌ Lỗi lấy danh sách coin từ Binance:
{str(e)}")
        return []

# Hàm get_klines (chỉ có trong v2_part1, cần thiết cho Scanner và các
chỉ báo phức tạp)
def get_klines(symbol, interval, limit=500):
    url = f"{BASE_FAPI}/fapi/v1/klines"
    params = {"symbol": symbol.upper(), "interval": interval, "limit":
limit}
    data = binance_api_request(url, params=params)

```

```

if not data: return None
# Trả về list 5 thành phần: open, high, low, close, volume
o, h, l, c, v = [], [], [], [], []
for line in data:
    try:
        o.append(float(line[1]))
        h.append(float(line[2]))
        l.append(float(line[3]))
        c.append(float(line[4]))
        v.append(float(line[5]))
    except (TypeError, ValueError):
        continue
return o, h, l, c, v

# Hàm get_qualified_symbols nâng cao từ v2_part2 (sử dụng logic
scoring)
def get_qualified_symbols_v2(api_key, api_secret, strategy_type,
leverage,
                                threshold=None, volatility=None,
grid_levels=None,
                                max_candidates=20, final_limit=2,
strategy_key=None):
    # Tích hợp logic scoring từ v2_part2
    # Do các chỉ báo ATR, ADX, EMA, etc. chỉ có ở cuối tệp này,
    # hàm này sẽ được gọi sau khi các chỉ báo được định nghĩa.
    # TẠM THỜI GIỮ LẠI HÀM get_qualified_symbols TỪ V16.PY
    # (Hàm này tuy đơn giản hơn nhưng không bị lỗi phụ thuộc chỉ báo)
    return get_qualified_symbols(api_key, api_secret, strategy_type,
leverage,
                                threshold, volatility, grid_levels,
                                max_candidates, final_limit,
strategy_key)

# Hàm get_qualified_symbols gốc từ v16.py (logic đơn giản)
def get_qualified_symbols(api_key, api_secret, strategy_type,
leverage, threshold=None, volatility=None, grid_levels=None,
max_candidates=20, final_limit=2, strategy_key=None):
    """Tìm coin phù hợp từ TOÀN BỘ Binance - PHÂN BIỆT THEO CẤU
HÌNH"""
    # (Nội dung hàm này quá dài, được giữ nguyên ở cuối tệp)
    # ...
    try:
        test_balance = get_balance(api_key, api_secret)
        if test_balance is None:
            logger.error("❌ KHÔNG THỂ KẾT NỐI BINANCE")
            return []

        coin_manager = CoinManager()

```

```

all_symbols = get_all_usdt_pairs(limit=200)
if not all_symbols:
    logger.error("✗ Không lấy được danh sách coin từ
Binance")
    return []

url = f"{BASE_FAPI}/fapi/v1/ticker/24hr"
data = binance_api_request(url)
if not data:
    return []

ticker_dict = {ticker['symbol']: ticker for ticker in data if
'symbol' in ticker}

qualified_symbols = []

for symbol in all_symbols:
    if symbol not in ticker_dict: continue
    if symbol in ['BTCUSDT', 'ETHUSDT']: continue
    if strategy_key and
coin_manager.has_same_config_bot(symbol, strategy_key): continue

    ticker = ticker_dict[symbol]

    try:
        price_change = float(ticker.get('priceChangePercent',
0))

        abs_price_change = abs(price_change)
        volume = float(ticker.get('quoteVolume', 0))
        high_price = float(ticker.get('highPrice', 0))
        low_price = float(ticker.get('lowPrice', 0))

        price_range = ((high_price - low_price) / low_price) *
100 if low_price > 0 else 0

        # ĐIỀU KIỆN CHO TỪNG CHIẾN LƯỢC
        if strategy_type == "Reverse 24h":
            if abs_price_change >= (threshold or 15) and
volume > 1000000:
                score = abs_price_change * (volume / 1000000)
                qualified_symbols.append((symbol, score,
price_change))

            elif strategy_type == "Scalping":
                if abs_price_change >= (volatility or 2) and
volume > 2000000 and price_range >= 1.0:
                    qualified_symbols.append((symbol,

```



```

price_range))

        elif strategy_type == "Safe Grid":
            if 0.5 <= abs_price_change <= 8.0 and volume >
500000:
                qualified_symbols.append((symbol,
-abs(price_change - 3.0))) # Ưu tiên biến động gần 3%

        elif strategy_type == "Trend Following":
            if (1.0 <= abs_price_change <= 15.0 and volume >
1000000 and price_range >= 0.5):
                score = volume * abs_price_change
                qualified_symbols.append((symbol, score))

        elif strategy_type == "Smart Dynamic":
            if (1.0 <= abs_price_change <= 12.0 and volume >
1500000 and price_range >= 0.8):
                volume_score = min(volume / 5000000, 5)
                volatility_score = min(abs_price_change / 10,
3)

                score = volume_score + volatility_score
                qualified_symbols.append((symbol, score))

    except (ValueError, TypeError) as e:
        continue

    # SẮP XẾP VÀ CHỌN TOP CANDIDATES (phần này vẫn giữ nguyên
logic từ v16.py)
    # ... (Cần đảm bảo logic set_leverage và get_step_size hoạt
động)

    # Sắp xếp theo score
    if qualified_symbols:
        qualified_symbols.sort(key=lambda x: x[1], reverse=True)

    final_symbols = []
    for item in qualified_symbols[:max_candidates]:
        if len(final_symbols) >= final_limit:
            break

    symbol = item[0]

    try:
        # set_leverage và get_step_size (cần được định nghĩa
trước)
        if not set_leverage(symbol, leverage, api_key,
api_secret):
            continue

```

```

        step_size = get_step_size(symbol, api_key, api_secret)
        if step_size <= 0:
            continue
        final_symbols.append(symbol)
        logger.info(f"✅ {symbol}: phù hợp {strategy_type}")

    except Exception as e:
        logger.error(f"❌ Lỗi kiểm tra {symbol}: {str(e)}")
        continue

    # Backup system
    if not final_symbols:
        logger.warning(f"⚠️ {strategy_type}: không tìm thấy coin
phù hợp, sử dụng backup method")
        backup_symbols = []

    for symbol in all_symbols:
        ticker = ticker_dict.get(symbol)
        if not ticker: continue
        try:
            volume = float(ticker.get('quoteVolume', 0))
            price_change =
float(ticker.get('priceChangePercent', 0))
            abs_price_change = abs(price_change)

            if (volume > 3000000 and 0.5 <= abs_price_change
<= 10.0 and symbol not in ['BTCUSDT', 'ETHUSDT']):
                backup_symbols.append((symbol, volume,
abs_price_change))
        except: continue

    backup_symbols.sort(key=lambda x: x[1], reverse=True)

    for symbol, _, _ in backup_symbols[:final_limit]:
        try:
            if len(final_symbols) >= final_limit: break
            if set_leverage(symbol, leverage, api_key,
api_secret) and get_step_size(symbol, api_key, api_secret) > 0:
                final_symbols.append(symbol)
                logger.info(f"🔄 {symbol}: backup coin")
        except: continue

    logger.info(f"🎯 {strategy_type}: Kết quả cuối -
{len(final_symbols)} coin: {final_symbols}")
    return final_symbols[:final_limit]

except Exception as e:
    logger.error(f"❌ Lỗi tìm coin {strategy_type}: {str(e)}")

```

```

        return []

def get_step_size(symbol, api_key, api_secret):
    # Hàm từ v2_part2/v16.py
    url = f"{BASE_FAPI}/fapi/v1/exchangeInfo"
    try:
        data = binance_api_request(url)
        if not data:
            return 0.001
        for s in data['symbols']:
            if s['symbol'] == symbol.upper():
                for f in s['filters']:
                    if f['filterType'] == 'LOT_SIZE':
                        return float(f['stepSize'])
    except Exception as e:
        logger.error(f"Lỗi lấy step size: {str(e)}")
    return 0.001

def set_leverage(symbol, lev, api_key, api_secret):
    # Hàm từ v2_part2/v16.py
    try:
        ts = int(time.time() * 1000)
        params = {
            "symbol": symbol.upper(),
            "leverage": lev,
            "timestamp": ts
        }
        query = urllib.parse.urlencode(params)
        sig = sign(query, api_secret)
        url = f"{BASE_FAPI}/fapi/v1/leverage?{query}&signature={sig}"
        headers = {'X-MBX-APIKEY': api_key}

        response = binance_api_request(url, method='POST',
headers=headers)
        if response is None:
            return False
        if response and 'leverage' in response:
            return True
        return False
    except Exception as e:
        logger.error(f"Lỗi thiết lập đòn bẩy: {str(e)}")
        return False

def get_balance(api_key, api_secret):
    # Hàm từ v2_part2/v16.py (chú ý v16.py chỉ lấy availableBalance,
v2_part2 lấy balance/availableBalance)
    try:
        ts = int(time.time() * 1000)

```

```

        params = {"timestamp": ts}
        query = urllib.parse.urlencode(params)
        sig = sign(query, api_secret)
        # Dùng v2/account để lấy vị thế, nhưng v2/balance cho số dư
        trực tiếp
        # Giữ nguyên logic của v16.py để đảm bảo tính nhất quán (lấy
        balance từ v2/account)
        url = f"{BASE_FAPI}/fapi/v2/account?{query}&signature={sig}"
        headers = {'X-MBX-APIKEY': api_key}

        data = binance_api_request(url, headers=headers)
        if not data:
            return None
        # Lấy availableBalance từ assets
        for asset in data['assets']:
            if asset['asset'] == 'USDT':
                return float(asset['availableBalance'])
        return 0
    except Exception as e:
        logger.error(f"Lỗi lấy số dư: {str(e)}")
        return None

def place_order(symbol, side, qty, api_key, api_secret):
    # Hàm từ v2_part2/v16.py
    try:
        ts = int(time.time() * 1000)
        params = {
            "symbol": symbol.upper(),
            "side": side,
            "type": "MARKET",
            "quantity": qty,
            "timestamp": ts
        }
        query = urllib.parse.urlencode(params)
        sig = sign(query, api_secret)
        url = f"{BASE_FAPI}/fapi/v1/order?{query}&signature={sig}"
        headers = {'X-MBX-APIKEY': api_key}

        return binance_api_request(url, method='POST',
        headers=headers)
    except Exception as e:
        logger.error(f"Lỗi đặt lệnh: {str(e)}")
        return None

def cancel_all_orders(symbol, api_key, api_secret):
    # Hàm từ v2_part2/v16.py
    try:
        ts = int(time.time() * 1000)

```

```

        params = {"symbol": symbol.upper(), "timestamp": ts}
        query = urllib.parse.urlencode(params)
        sig = sign(query, api_secret)
        url =
f"{BASE_FAPI}/fapi/v1/allOpenOrders?{query}&signature={sig}"
        headers = {'X-MBX-APIKEY': api_key}

        binance_api_request(url, method='DELETE', headers=headers)
        return True
    except Exception as e:
        logger.error(f"Lỗi hủy lệnh: {str(e)}")
    return False

def get_current_price(symbol):
    # Hàm từ v2_part2/v16.py
    try:
        url =
f"{BASE_FAPI}/fapi/v1/ticker/price?symbol={symbol.upper()}"
        data = binance_api_request(url)
        if data and 'price' in data:
            return float(data['price'])
    except Exception as e:
        logger.error(f"Lỗi lấy giá: {str(e)}")
    return 0

def get_positions(symbol=None, api_key=None, api_secret=None):
    # Hàm từ v16.py
    try:
        ts = int(time.time() * 1000)
        params = {"timestamp": ts}
        query = urllib.parse.urlencode(params)
        sig = sign(query, api_secret)
        # Sử dụng fapi/v2/account để lấy vị thế
        url = f"{BASE_FAPI}/fapi/v2/account?{query}&signature={sig}"
        headers = {'X-MBX-APIKEY': api_key}

        data = binance_api_request(url, headers=headers)
        if not data:
            return []

        positions = []
        for pos in data.get('positions', []):
            if float(pos.get('positionAmt', 0)) != 0:
                if symbol and pos.get('symbol') != symbol.upper():
                    continue
                positions.append({
                    'symbol': pos.get('symbol'),
                    'positionAmt': float(pos.get('positionAmt', 0)),

```

```

        'entryPrice': float(pos.get('entryPrice', 0)),
        'unRealizedProfit':
float(pos.get('unRealizedProfit', 0))
    })
    return positions
except Exception as e:
    logger.error(f"Lỗi lấy vị thế: {str(e)}")
return []

def get_24h_change(symbol):
    # Hàm từ v16.py
    try:
        url =
f"{BASE_FAPI}/fapi/v1/ticker/24hr?symbol={symbol.upper()}"
        data = binance_api_request(url)
        if data and 'priceChangePercent' in data:
            change = data['priceChangePercent']
            return float(change) if change is not None else 0.0
        return 0.0
    except Exception as e:
        logger.error(f"Lỗi lấy biến động 24h cho {symbol}: {str(e)}")
    return 0.0

# ===== CHỈ BÁO KỸ THUẬT =====
# Tích hợp toàn bộ chỉ báo từ v2_part1, giữ lại calc_rsi/calc_ema đơn
giản từ v16.py để fallback nếu cần.

def rsi_wilder_last(prices, period=14):
    if len(prices) < period + 1: return None
    deltas = np.diff(prices)
    gains = np.where(deltas > 0, deltas, 0.0)
    losses = np.where(deltas < 0, -deltas, 0.0)
    # Tính AVG ban đầu
    avg_gain = np.sum(gains[:period]) / period
    avg_loss = np.sum(losses[:period]) / period

    # Tính RSI đầu tiên
    rs = avg_gain / avg_loss if avg_loss != 0 else (9999 if avg_gain >
0 else 0)
    rsi = 100 - 100 / (1 + rs)

    # Áp dụng Wilder Smoothing cho các giá trị sau
    for i in range(period, len(deltas)):
        gain, loss = gains[i], losses[i]
        avg_gain = (avg_gain * (period - 1) + gain) / period
        avg_loss = (avg_loss * (period - 1) + loss) / period
        rs = avg_gain / avg_loss if avg_loss != 0 else (9999 if
avg_gain > 0 else 0)

```

```

        rsi = 100 - 100 / (1 + rs)

    return float(rsi)

# Alias cho calc_rsi đơn giản (dùng trong v16.py)
calc_rsi = rsi_wilder_last

def ema_last(values, period):
    if len(values) < period: return None
    k = 2/(period+1)
    ema = float(values[0])
    for v in values[1:]:
        ema = v*k + ema*(1-k)
    return float(ema)

# Alias cho calc_ema đơn giản (dùng trong v16.py)
calc_ema = ema_last

def atr_last(highs, lows, closes, period=14, return_pct=True):
    n = len(closes)
    if min(len(highs), len(lows), len(closes)) < period+1: return None
    trs = []
    for i in range(1, n):
        tr = max(highs[i]-lows[i], abs(highs[i]-closes[i-1]),
abs(lows[i]-closes[i-1]))
        trs.append(tr)

    # Wilder Smoothing
    def wilder(arr, p=period):
        a = sum(arr[:p]) / p
        for x in arr[p:]:
            a = (a*(p-1) + x) / p
        return a

    atr = wilder(trs)

    return float(atr / closes[-1] * 100.0) if return_pct and
closes[-1] > 0 else float(atr)

def adx_last(highs, lows, closes, period=14):
    n = len(closes)
    if min(len(highs), len(lows), len(closes)) < period+1: return
None, None, None

    plus_dm, minus_dm, trs = [], [], []
    for i in range(1, n):
        up = highs[i]-highs[i-1]
        down = lows[i-1]-lows[i]

```

```

        plus_dm.append(up if (up>down and up>0) else 0.0)
        minus_dm.append(down if (down>up and down>0) else 0.0)
        tr = max(highs[i]-lows[i], abs(highs[i]-closes[i-1]),
abs(lows[i]-closes[i-1]))
        trs.append(tr)

def wilder(arr, p=period):
    if len(arr) < p: return 0.0
    a = sum(arr[:p]) / p
    for x in arr[p:]:
        a = (a*(p-1) + x) / p
    return a

atr = wilder(trs)
if atr == 0: return None, None, None

plus_di = 100 * (wilder(plus_dm) / atr)
minus_di = 100 * (wilder(minus_dm) / atr)

dx = 100 * abs(plus_di - minus_di) / max(plus_di + minus_di, 1e-9)
# ADX là Wilder Smoothing của DX

# Cần tính ADX một cách đúng đắn, ở đây chỉ tính DX cuối cùng
# Do logic này quá phức tạp nếu không có thư viện TA, ta giữ
nguyên ADX = DX ở đây
adx = dx

return float(adx), float(plus_di), float(minus_di)

def bbands_last(closes, period=20, std=2):
    if len(closes) < period: return None, None, None
    w = np.array(closes[-period:])
    mid = w.mean(); dev = w.std(ddof=0)
    return float(mid), float(mid+std*dev), float(mid-std*dev)

def mfi_last(highs, lows, closes, volumes, period=14):
    n = len(closes)
    if min(len(highs), len(lows), len(closes), len(volumes)) <
period+1: return None
    tp = (np.array(highs) + np.array(lows) + np.array(closes)) / 3.0
    raw = tp * np.array(volumes)
    pos, neg = [], []
    for i in range(1, n):
        if tp[i] > tp[i-1]: pos.append(raw[i]); neg.append(0.0)
        elif tp[i] < tp[i-1]: pos.append(0.0); neg.append(raw[i])
        else: pos.append(0.0); neg.append(0.0)

    if len(pos) < period: return None

```



```

# Tính sum cho 14 kỳ gần nhất (tính từ len(pos)-1 trở về trước)
p = sum(pos[-period:]); q = sum(neg[-period:])

if q == 0: return 100.0

mr = p/q
return float(100 - 100/(1+mr))

def obv_last(closes, volumes):
    if len(closes) < 2 or len(volumes) < 2: return 0.0
    obv = 0.0
    for i in range(1, len(closes)):
        if closes[i] > closes[i-1]: obv += volumes[i]
        elif closes[i] < closes[i-1]: obv -= volumes[i]
    return float(obv)

# ===== WEBSOCKET MANAGER =====
class WebSocketManager:
    # Giữ nguyên bản từ v16.py (sử dụng websocket thực)
    def __init__(self):
        self.connections = {}
        self.executor = ThreadPoolExecutor(max_workers=10)
        self._lock = threading.Lock()
        self._stop_event = threading.Event()

    def add_symbol(self, symbol, callback):
        symbol = symbol.upper()
        with self._lock:
            if symbol not in self.connections:
                self._create_connection(symbol, callback)

    def _create_connection(self, symbol, callback):
        if self._stop_event.is_set():
            return
        stream = f"{symbol.lower()}@trade"
        url = f"wss://fstream.binance.com/ws/{stream}"

    def on_message(ws, message):
        try:
            data = json.loads(message)
            if 'p' in data:
                price = float(data['p'])
                self.executor.submit(callback, price)
        except Exception as e:
            logger.error(f"Lỗi xử lý tin nhắn WebSocket {symbol}: {str(e)}")

```

```

def on_error(ws, error):
    logger.error(f"Lỗi WebSocket {symbol}: {str(error)}")
    if not self._stop_event.is_set():
        time.sleep(5)
        self._reconnect(symbol, callback)

def on_close(ws, close_status_code, close_msg):
    logger.info(f"WebSocket đóng {symbol}: {close_status_code}
- {close_msg}")
    if not self._stop_event.is_set() and symbol in
self.connections:
        time.sleep(5)
        self._reconnect(symbol, callback)

ws = websocket.WebSocketApp(
    url,
    on_message=on_message,
    on_error=on_error,
    on_close=on_close
)

thread = threading.Thread(target=ws.run_forever, daemon=True)
thread.start()

self.connections[symbol] = {
    'ws': ws,
    'thread': thread,
    'callback': callback
}
logger.info(f"WebSocket bắt đầu cho {symbol}")

def _reconnect(self, symbol, callback):
    logger.info(f"Kết nối lại WebSocket cho {symbol}")
    self.remove_symbol(symbol)
    self._create_connection(symbol, callback)

def remove_symbol(self, symbol):
    symbol = symbol.upper()
    with self._lock:
        if symbol in self.connections:
            try:
                self.connections[symbol]['ws'].close()
            except Exception as e:
                logger.error(f"Lỗi đóng WebSocket {symbol}:
{str(e)}")

            del self.connections[symbol]
            logger.info(f"WebSocket đã xóa cho {symbol}")

```

```

def stop(self):
    self._stop_event.set()
    for symbol in list(self.connections.keys()):
        self.remove_symbol(symbol)

# ===== BASE BOT NÂNG CẤP VỚI TÍNH NĂNG TÌM COIN MỚI =====
class BaseBot:
    def __init__(self, symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret,
        telegram_bot_token, telegram_chat_id, strategy_name,
config_key=None,
        smart_exit_config=None, dynamic_mode=False):

        self.symbol = symbol.upper() if symbol else "BTCUSDT"
        self.lev = lev
        self.percent = percent
        self.tp = tp
        self.sl = sl
        self.ws_manager = ws_manager
        self.api_key = api_key
        self.api_secret = api_secret
        self.telegram_bot_token = telegram_bot_token
        self.telegram_chat_id = telegram_chat_id
        self.strategy_name = strategy_name
        self.config_key = config_key
        self.dynamic_mode = dynamic_mode

        self.status = "waiting"
        self.side = ""
        self.qty = 0
        self.entry = 0
        self.prices = [] # Dùng cho chỉ báo
        self.position_open = False
        self._stop = False

        self.last_trade_time = 0
        self.last_close_time = 0
        self.last_position_check = 0
        self.last_error_log_time = 0

        self.cooldown_period = 300
        self.position_check_interval = 30

        self._close_attempted = False
        self._last_close_attempt = 0
        self.should_be_removed = False # Cờ để BotManager xóa bot

        self.coin_manager = CoinManager()

```

```

# HỆ THỐNG SMART EXIT
self.smart_exit = SmartExitManager(self)
if smart_exit_config:
    self.smart_exit.update_config(**smart_exit_config)

# Đăng ký coin ban đầu
if symbol:
    success = self.coin_manager.register_coin(self.symbol,
f"{strategy_name}_{id(self)}", strategy_name, config_key)
    if not success:
        self.log(f"⚠ Cảnh báo: {self.symbol} đã được quản lý
bởi bot khác hoặc đang cooldown")

# Khởi tạo và chạy thread
self.check_position_status()
self.ws_manager.add_symbol(self.symbol,
self._handle_price_update)
self.thread = threading.Thread(target=self._run, daemon=True)
self.thread.start()

mode_text = "ĐỘNG - Tự tìm coin" if dynamic_mode else "TĨNH -
Coin cố định"
self.log(f"🟢 Bot {strategy_name} khởi động | {self.symbol} |
Chế độ: {mode_text} | ĐB: {lev}x | Vốn: {percent}% | TP/SL:
{tp}%/{sl}%")

def log(self, message):
    logger.info(f"[{self.symbol} - {self.strategy_name}]
{message}")
    if self.telegram_bot_token and self.telegram_chat_id:
        send_telegram(f"<b>{self.symbol}</b>
({self.strategy_name}): {message}",
bot_token=self.telegram_bot_token,
default_chat_id=self.telegram_chat_id)

def _handle_price_update(self, price):
    if self._stop or not price or price <= 0:
        return
    try:
        self.prices.append(float(price))
        if len(self.prices) > 1000: # Tăng buffer để đủ cho các
chỉ báo
            self.prices = self.prices[-1000:]
    except Exception as e:
        self.log(f"❌ Lỗi xử lý giá: {str(e)}")

def get_signal(self):

```

```

        # Đây là phương thức trừu tượng, sẽ được triển khai trong lớp
con
        return None

    def check_position_status(self):
        # ... (Nội dung giữ nguyên từ v16.py)
        try:
            positions = get_positions(self.symbol, self.api_key,
self.api_secret)
            if not positions:
                self._reset_position()
                return

            position_found = False
            for pos in positions:
                if pos['symbol'] == self.symbol:
                    position_amt = float(pos.get('positionAmt', 0))
                    if abs(position_amt) > 0:
                        position_found = True
                        self.position_open = True
                        self.status = "open"
                        self.side = "BUY" if position_amt > 0 else
"SELL"

                        self.qty = position_amt
                        self.entry = float(pos.get('entryPrice', 0))
                        break
                    else:
                        position_found = True
                        self._reset_position()
                        break

            if not position_found:
                self._reset_position()

        except Exception as e:
            if time.time() - self.last_error_log_time > 10:
                self.log(f"❌ Lỗi kiểm tra vị thế: {str(e)}")
                self.last_error_log_time = time.time()

    def _reset_position(self):
        self.position_open = False
        self.status = "waiting"
        self.side = ""
        self.qty = 0
        self.entry = 0
        self._close_attempted = False
        self._last_close_attempt = 0

```

```

        self.smart_exit.trailing_active = False # Reset smart exit
state
        self.smart_exit._breakeven_active = False
        self.smart_exit._tp_hit.clear()

    def _run(self):
        # ... (Nội dung giữ nguyên từ v16.py)
        while not self._stop:
            try:
                current_time = time.time()

                if current_time - self.last_position_check >
self.position_check_interval:
                    self.check_position_status()
                    self.last_position_check = current_time

                if self.should_be_removed:
                    self.log("🔴 Bot đã được đánh dấu xóa, dừng hoạt
động")

                    time.sleep(1)
                    continue

                if not self.position_open:
                    signal = self.get_signal()

                    if (signal and
                        current_time - self.last_trade_time > 60 and
                        current_time - self.last_close_time >
self.cooldown_period and
                        not self.should_be_removed):

                        self.log(f"🔴 Nhận tín hiệu {signal}, đang mở
lệnh...")

                        if self.open_position(signal):
                            self.last_trade_time = current_time
                        else:
                            time.sleep(30)

                        if self.position_open and not self._close_attempted
and not self.should_be_removed:
                            self.check_tp_sl()

                            time.sleep(1)

            except Exception as e:
                if time.time() - self.last_error_log_time > 10:
                    self.log(f"❌ Lỗi hệ thống: {str(e)}")
                    self.last_error_log_time = time.time()

```

```

        time.sleep(1)

def stop(self):
    self._stop = True
    self.ws_manager.remove_symbol(self.symbol)
    self.coin_manager.unregister_coin(self.symbol)
    cancel_all_orders(self.symbol, self.api_key, self.api_secret)
    self.log(f"🛑 Bot dừng cho {self.symbol}")

def open_position(self, side):
    # ... (Nội dung giữ nguyên từ v16.py)
    try:
        self.check_position_status()
        if self.position_open:
            self.log(f"⚠️ Đã có vị thế {self.side}, bỏ qua tín
hiệu {side}")
            return False

        if self.should_be_removed:
            self.log(f"⚠️ Bot đã được đánh dấu xóa, không mở lệnh
mới")
            return False

        if not set_leverage(self.symbol, self.lev, self.api_key,
self.api_secret):
            self.log(f"❌ Không thể đặt đòn bẩy {self.lev}x")
            return False

        balance = get_balance(self.api_key, self.api_secret)
        if balance is None or balance <= 0:
            self.log(f"❌ Không đủ số dư")
            return False

        current_price = get_current_price(self.symbol)
        if current_price <= 0:
            self.log(f"❌ Lỗi lấy giá")
            return False

        step_size = get_step_size(self.symbol, self.api_key,
self.api_secret)
        usd_amount = balance * (self.percent / 100)
        qty = (usd_amount * self.lev) / current_price

        if step_size > 0:
            # Tính toán lại qty theo step_size
            qty = math.floor(qty / step_size) * step_size
            # Đảm bảo precision
            precision = int(round(-math.log10(step_size))) if

```

```

step_size < 1 else 0
    qty = float(f"{qty:.{precision}f}")

    if qty < step_size:
        self.log(f"❌ Số lượng quá nhỏ: {qty}")
        return False

    result = place_order(self.symbol, side, qty, self.api_key,
self.api_secret)
    if result and 'orderId' in result:
        executed_qty = float(result.get('executedQty', 0))
        avg_price = float(result.get('avgPrice',
current_price))

        if executed_qty > 0:
            self.entry = avg_price
            self.side = side
            self.qty = executed_qty if side == "BUY" else
-executed_qty

            self.position_open = True
            self.status = "open"

            self.smart_exit.on_position_opened()

            message = (
                f"✅ <b>ĐÃ MỞ VỊ THẾ {self.symbol}</b>\n"
                f"🤖 Chiến lược: {self.strategy_name}\n"
                f"📌 Hướng: {side}\n"
                f"👉 Giá vào: {self.entry:.4f}\n"
                f"📊 Khối lượng: {executed_qty:.4f}\n"
                f"💵 Giá trị: {executed_qty * self.entry:.2f}
USDT\n"

                f"💰 Đòn bẩy: {self.lev}x\n"
                f"🎯 TP: {self.tp}% | 🛡️ SL: {self.sl}%"
            )
            self.log(message)
            return True
        else:
            self.log(f"❌ Lệnh không khớp - Số lượng: {qty}")
            return False
    else:
        error_msg = result.get('msg', 'Unknown error') if
result else 'No response'
        self.log(f"❌ Lỗi đặt lệnh {side}: {error_msg}")
        return False

except Exception as e:
    self.log(f"❌ Lỗi mở lệnh: {str(e)}")

```



```

        return False

def close_position(self, reason=""):
    # ... (Nội dung giữ nguyên từ v16.py)
    if not self.position_open or self._close_attempted:
        return False

    current_time = time.time()
    if self._close_attempted and current_time -
self._last_close_attempt < 30:
        self.log(f"⚠ Đang thử đóng lệnh lần trước, chờ...")
        return False

    try:
        self._close_attempted = True
        self._last_close_attempt = current_time

        close_side = "SELL" if self.side == "BUY" else "BUY"
        close_qty = abs(self.qty)

        # Hủy lệnh đang mở và chờ
        cancel_all_orders(self.symbol, self.api_key,
self.api_secret)
        time.sleep(0.5)

        # Đóng lệnh Market
        result = place_order(self.symbol, close_side, close_qty,
self.api_key, self.api_secret)

        if result and 'orderId' in result:
            current_price = get_current_price(self.symbol)
            pnl = 0
            if self.entry > 0 and current_price > 0:
                if self.side == "BUY":
                    pnl = (current_price - self.entry) *
abs(self.qty)
                else:
                    pnl = (self.entry - current_price) *
abs(self.qty)

            message = (
                f"🛑 <b>ĐÃ ĐÓNG VỊ THẾ {self.symbol}</b>\n"
                f"🤖 Chiến lược: {self.strategy_name}\n"
                f"📌 Lý do: {reason}\n"
                f"🏷 Giá ra: {current_price:.4f}\n"
                f"📊 Khối lượng: {close_qty:.4f}\n"
                f"💰 PnL: {pnl:.2f} USDT"
            )

```

```

        self.log(message)

        # Set cooldown cho coin cũ
        old_symbol = self.symbol
        self.coin_manager.set_cooldown(old_symbol)
        self.log(f"🕒 COOLDOWN {old_symbol}
({self.coin_manager.cooldown_left(old_symbol)}s)")

        # Bot Động tìm coin mới
        if self.dynamic_mode:
            self.log("🔄 Bot động: Đang tìm coin mới...")

threading.Thread(target=self._find_new_coin_after_exit,
daemon=True).start()
        else:
            self.should_be_removed = True # BotManager sẽ xóa
bot tĩnh

        self._reset_position()
        self.last_close_time = time.time()

        time.sleep(2)
        self.check_position_status()

        return True
    else:
        error_msg = result.get('msg', 'Unknown error') if
result else 'No response'
        self.log(f"❌ Lỗi đóng lệnh: {error_msg}")
        self._close_attempted = False
        return False

    except Exception as e:
        self.log(f"❌ Lỗi đóng lệnh: {str(e)}")
        self._close_attempted = False
        return False

def _find_new_coin_after_exit(self):
    """🔄 TÌM COIN MỚI CHO BOT ĐỘNG SAU KHI ĐÓNG LỆNH"""
    try:
        self.log("🔄 Bot động đang tìm coin mới...")

        # Hàm get_qualified_symbols từ v16.py (đã được tích hợp ở
trên)

        new_symbols = get_qualified_symbols(
            self.api_key,
            self.api_secret,
            self.strategy_name,

```

```

        self.lev,
        threshold=getattr(self, 'threshold', None),
        volatility=getattr(self, 'volatility', None),
        grid_levels=getattr(self, 'grid_levels', None),
        max_candidates=10,
        final_limit=1,
        strategy_key=self.config_key
    )

    if new_symbols:
        new_symbol = new_symbols[0]

        # Hủy đăng ký coin cũ
        old_symbol = self.symbol
        self.coin_manager.unregister_coin(old_symbol)

        # Cập nhật symbol mới
        self.symbol = new_symbol

        # Đăng ký coin mới (sẽ thất bại nếu coin mới đang
cooldown)
        registered = self.coin_manager.register_coin(
            new_symbol, f"{self.strategy_name}_{id(self)}",
self.strategy_name, self.config_key
        )

        if registered:
            self._restart_websocket_for_new_coin()

            message = f"🔄 Bot động chuyển từ {old_symbol} →
{new_symbol}"

            self.log(message)

            self.should_be_removed = False
        else:
            self.log(f"❌ Không thể đăng ký coin mới
{new_symbol} (có thể đang cooldown)")
            # Quay lại coin cũ nếu không đăng ký được
            self.symbol = old_symbol
            self.coin_manager.register_coin(old_symbol,
f"{self.strategy_name}_{id(self)}", self.strategy_name,
self.config_key)
        else:
            self.log(f"❌ Không tìm thấy coin mới phù hợp, giữ
nguyên coin hiện tại")

    except Exception as e:
        self.log(f"❌ Lỗi tìm coin mới: {str(e)}")

```

```

        traceback.print_exc()

def _restart_websocket_for_new_coin(self):
    """Khởi động lại WebSocket cho coin mới"""
    try:
        self.ws_manager.remove_symbol(self.symbol)
        time.sleep(2)
        self.ws_manager.add_symbol(self.symbol,
self._handle_price_update)
        self.log(f"🔗 Khởi động lại WebSocket cho {self.symbol}")

    except Exception as e:
        self.log(f"❌ Lỗi khởi động lại WebSocket: {str(e)}")

def check_tp_sl(self):
    """KIỂM TRA SMART EXIT + TP/SL TRUYỀN THỐNG"""

    # 1. KIỂM TRA SMART EXIT TRƯỚC
    if self.position_open and self.entry > 0:
        current_price = get_current_price(self.symbol)
        if current_price > 0:
            # Logic volume check cần giá trị volume, ở đây không
có nên chỉ check giá
            exit_reason =
self.smart_exit.check_all_exit_conditions(current_price)
            if exit_reason:
                self.close_position(exit_reason)
            return

    # 2. KIỂM TRA TP/SL TRUYỀN THỐNG
    if not self.position_open or self.entry <= 0 or
self._close_attempted:
        return

    current_price = get_current_price(self.symbol)
    if current_price <= 0:
        return

    # Tính ROI
    if self.side == "BUY":
        profit = (current_price - self.entry) * abs(self.qty)
    else:
        profit = (self.entry - current_price) * abs(self.qty)

    invested = self.entry * abs(self.qty) / self.lev
    if invested <= 0:
        return

```

```

    roi = (profit / invested) * 100

    if self.tp is not None and roi >= self.tp:
        self.close_position(f"✅ Đạt TP {self.tp}% (ROI:
{roi:.2f}%)")
    elif self.sl is not None and self.sl > 0 and roi <= -self.sl:
        self.close_position(f"❌ Đạt SL {self.sl}% (ROI:
{roi:.2f}%)")

# ===== CÁC CHIẾN LƯỢC GIAO DỊCH =====
class RSI_EMA_Bot(BaseBot):
    # Dùng chỉ báo calc_rsi (rsi_wilder_last) và calc_ema (ema_last)
    def __init__(self, symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret, telegram_bot_token, telegram_chat_id,
smart_exit_config=None, dynamic_mode=False):
        super().__init__(symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret, telegram_bot_token, telegram_chat_id, "RSI/EMA
Recursive", smart_exit_config=smart_exit_config,
dynamic_mode=dynamic_mode)
        self.rsi_period = 14
        self.ema_fast = 9
        self.ema_slow = 21
        self.rsi_oversold = 30
        self.rsi_overbought = 70

    def get_signal(self):
        try:
            if len(self.prices) < 50:
                return None

            # Dùng calc_rsi/calc_ema
            rsi = calc_rsi(self.prices, self.rsi_period)
            ema_fast = calc_ema(self.prices, self.ema_fast)
            ema_slow = calc_ema(self.prices, self.ema_slow)

            if rsi is None or ema_fast is None or ema_slow is None:
                return None

            signal = None
            if rsi < self.rsi_oversold and ema_fast > ema_slow:
                signal = "BUY"
            elif rsi > self.rsi_overbought and ema_fast < ema_slow:
                signal = "SELL"

            return signal

        except Exception as e:
            return None

```

```

class EMA_Crossover_Bot(BaseBot):
    # Dùng chỉ báo calc_ema (ema_last)
    def __init__(self, symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret, telegram_bot_token, telegram_chat_id,
smart_exit_config=None, dynamic_mode=False):
        super().__init__(symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret, telegram_bot_token, telegram_chat_id, "EMA
Crossover", smart_exit_config=smart_exit_config,
dynamic_mode=dynamic_mode)
        self.ema_fast = 9
        self.ema_slow = 21
        self.prev_ema_fast = None
        self.prev_ema_slow = None

    def get_signal(self):
        try:
            if len(self.prices) < 50:
                return None

            ema_fast = calc_ema(self.prices, self.ema_fast)
            ema_slow = calc_ema(self.prices, self.ema_slow)

            if ema_fast is None or ema_slow is None:
                return None

            signal = None
            if self.prev_ema_fast is not None and self.prev_ema_slow
is not None:
                if self.prev_ema_fast <= self.prev_ema_slow and
ema_fast > ema_slow:
                    signal = "BUY"
                elif self.prev_ema_fast >= self.prev_ema_slow and
ema_fast < ema_slow:
                    signal = "SELL"

            self.prev_ema_fast = ema_fast
            self.prev_ema_slow = ema_slow

            return signal

        except Exception as e:
            return None

class Reverse_24h_Bot(BaseBot):
    # Dùng hàm get_24h_change
    def __init__(self, symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret, telegram_bot_token, telegram_chat_id,

```


```

threshold=30, config_key=None, smart_exit_config=None,
dynamic_mode=False):
    super().__init__(symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret, telegram_bot_token, telegram_chat_id, "Reverse
24h", config_key, smart_exit_config=smart_exit_config,
dynamic_mode=dynamic_mode)
    self.threshold = threshold
    self.last_24h_check = 0
    self.last_reported_change = 0

def get_signal(self):
    try:
        current_time = time.time()
        if current_time - self.last_24h_check < 60: # Kiểm tra mỗi
60s
            return None


        change_24h = get_24h_change(self.symbol)
        self.last_24h_check = current_time

        if change_24h is None:
            return None

        if abs(change_24h - self.last_reported_change) > 5:
            self.log(f" Biến động 24h: {change_24h:.2f}% |
Ngưỡng: {self.threshold}%")
            self.last_reported_change = change_24h

        signal = None
        if abs(change_24h) >= self.threshold:
            if change_24h > 0:
                signal = "SELL"
            else:
                signal = "BUY"

        return signal

    except Exception as e:
        self.log(f" Lỗi tín hiệu Reverse 24h: {str(e)}")
        return None

class Trend_Following_Bot(BaseBot):
    def __init__(self, symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret, telegram_bot_token, telegram_chat_id,
config_key=None, smart_exit_config=None, dynamic_mode=False):
        super().__init__(symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret, telegram_bot_token, telegram_chat_id, "Trend
Following", config_key, smart_exit_config=smart_exit_config,

```

```

dynamic_mode=dynamic_mode)
    self.trend_period = 20
    self.trend_threshold = 0.001

    def get_signal(self):
        try:
            if len(self.prices) < self.trend_period + 1:
                return None

            recent_prices = self.prices[-self.trend_period:]
            if len(recent_prices) < 2:
                return None

            # Tính phần trăm thay đổi trong trend_period
            price_change = (recent_prices[-1] - recent_prices[0]) /
recent_prices[0]

            signal = None
            if price_change > self.trend_threshold:
                signal = "BUY"
            elif price_change < -self.trend_threshold:
                signal = "SELL"

            return signal

        except Exception as e:
            return None

class Scalping_Bot(BaseBot):
    # Dùng chỉ báo calc_rsi (rsi_wilder_last)
    def __init__(self, symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret, telegram_bot_token, telegram_chat_id,
config_key=None, smart_exit_config=None, dynamic_mode=False):
        super().__init__(symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret, telegram_bot_token, telegram_chat_id, "Scalping",
config_key, smart_exit_config=smart_exit_config,
dynamic_mode=dynamic_mode)
        self.rsi_period = 7
        self.min_movement = 0.001

    def get_signal(self):
        try:
            if len(self.prices) < 20:
                return None

            current_price = self.prices[-1]
            price_change = 0
            if len(self.prices) >= 2:

```



```

        price_change = (current_price - self.prices[-2]) /
self.prices[-2]

        rsi = calc_rsi(self.prices, self.rsi_period)

        if rsi is None:
            return None

        signal = None
        if rsi < 25 and price_change < -self.min_movement: # Quá
bán + giảm mạnh
            signal = "BUY"
        elif rsi > 75 and price_change > self.min_movement: # Quá
mua + tăng mạnh
            signal = "SELL"

        return signal

    except Exception as e:
        return None

class Safe_Grid_Bot(BaseBot):
    def __init__(self, symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret, telegram_bot_token, telegram_chat_id,
grid_levels=5, config_key=None, smart_exit_config=None,
dynamic_mode=False):
        super().__init__(symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret, telegram_bot_token, telegram_chat_id, "Safe
Grid", config_key, smart_exit_config=smart_exit_config,
dynamic_mode=dynamic_mode)
        self.grid_levels = grid_levels
        self.orders_placed = 0

    def get_signal(self):
        try:
            # Logic Grid giả định: mở lệnh BUY/SELL xen kẽ cho đến khi
đạt grid_levels
            if self.orders_placed < self.grid_levels:
                self.orders_placed += 1
                if self.orders_placed % 2 == 1:
                    return "BUY"
                else:
                    return "SELL"
            return None
        except Exception as e:
            return None

# ===== BOT ĐỘNG THÔNG MINH =====

```

```

class SmartDynamicBot(BaseBot):
    """BOT ĐỘNG THÔNG MINH - KẾT HỢP NHIỀU CHIẾN LƯỢC"""

    def __init__(self, symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret,
        telegram_bot_token, telegram_chat_id,
config_key=None, smart_exit_config=None, dynamic_mode=True):

        super().__init__(symbol, lev, percent, tp, sl, ws_manager,
api_key, api_secret,
        telegram_bot_token, telegram_chat_id, "Smart
Dynamic", config_key, smart_exit_config, dynamic_mode)

        # KÍCH HOẠT SMART EXIT MẶC ĐỊNH
        default_smart_config = {
            'enable_trailing': True,
            'enable_time_exit': True,
            'enable_support_resistance': True,
            'trailing_activation': 30,
            'trailing_distance': 15,
            'max_hold_time': 4,
            'min_profit_for_exit': 15
        }
        self.smart_exit.update_config(**default_smart_config)

    def get_signal(self):
        """KẾT HỢP NHIỀU CHIẾN LƯỢC ĐỂ RA TÍN HIỆU TỐI ƯU"""
        try:
            if len(self.prices) < 50:
                return None

            # 1. RSI SIGNAL
            rsi = calc_rsi(self.prices, 14)

            # 2. EMA SIGNAL
            ema_fast = calc_ema(self.prices, 9)
            ema_slow = calc_ema(self.prices, 21)

            # 3. TREND SIGNAL
            trend_strength = self._calculate_trend_strength()

            # 4. VOLATILITY CHECK
            volatility = self._calculate_volatility()

            if None in [rsi, ema_fast, ema_slow]:
                return None

            signal = None

```

```

        score = 0

        # RSI + EMA CONFIRMATION
        if rsi < 30 and ema_fast > ema_slow:
            score += 2
            signal = "BUY"
        elif rsi > 70 and ema_fast < ema_slow:
            score += 2
            signal = "SELL"

        # TREND CONFIRMATION
        if trend_strength > 0.005 and signal == "BUY": # Trend
strength là %
            score += 1
        elif trend_strength < -0.005 and signal == "SELL":
            score += 1

        # VOLATILITY FILTER (tránh market quá biến động)
        if volatility > 8.0:
            score -= 1

        # CHỈ VÀO LỆNH KHI SCORE ĐỦ CAO
        if score >= 2:
            self.log(f"🎯 Smart Signal: {signal} | Score:
{score}/3 | RSI: {rsi:.1f} | Trend: {trend_strength*100:.2f}%")
            return signal

        return None

    except Exception as e:
        self.log(f"❌ Lỗi Smart Dynamic signal: {str(e)}")
        return None

    def _calculate_trend_strength(self):
        """Tính strength của trend (trả về giá trị % thay đổi)"""
        if len(self.prices) < 20:
            return 0

        # Lấy giá trị đầu và cuối của 20 giá gần nhất
        p_start = self.prices[-20]
        p_end = self.prices[-1]

        # Trả về % thay đổi
        if p_start > 0:
            return (p_end - p_start) / p_start
        return 0

    def _calculate_volatility(self):

```

```

        """Tính độ biến động (trả về trung bình % thay đổi giữa các
n ền)"""
        if len(self.prices) < 20:
            return 0

        returns = []
        for i in range(len(self.prices)-20, len(self.prices)):
            if i > 0 and self.prices[i-1] > 0:
                ret = (self.prices[i] - self.prices[i-1]) /
self.prices[i-1]
                returns.append(abs(ret))

        return np.mean(returns) * 100

# ===== BOT MANAGER HOÀN CHỈNH VỚI TÍNH NĂNG BOT ĐỘNG =====
class BotManager:
    # Class này chứa toàn bộ logic quản lý bot, auto-scan, telegram
listener
    # ... (Nội dung lớp BotManager từ v16.py)
    def __init__(self, api_key=None, api_secret=None,
telegram_bot_token=None, telegram_chat_id=None):
        self.ws_manager = WebSocketManager()
        self.bots = {}
        self.running = True
        self.start_time = time.time()
        self.user_states = {}

        self.auto_strategies = {}
        self.last_auto_scan = 0
        self.auto_scan_interval = 600

        self.strategy_cooldowns = {
            "Reverse 24h": {},
            "Scalping": {},
            "Trend Following": {},
            "Safe Grid": {},
            "Smart Dynamic": {}
        }
        self.cooldown_period = 300 # 5 phút cooldown

        self.api_key = api_key
        self.api_secret = api_secret
        self.telegram_bot_token = telegram_bot_token
        self.telegram_chat_id = telegram_chat_id

        if api_key and api_secret:
            self._verify_api_connection()
            self.log("🟢 HỆ THỐNG BOT THÔNG MINH ĐÃ KHỞI ĐỘNG")

```

```

        # Chỉ khởi động listener nếu có chat_id
        if self.telegram_chat_id:
            self.telegram_thread =
threading.Thread(target=self._telegram_listener, daemon=True)
            self.telegram_thread.start()
            self.send_main_menu(self.telegram_chat_id)

        self.auto_scan_thread =
threading.Thread(target=self._auto_scan_loop, daemon=True)
        self.auto_scan_thread.start()

    else:
        self.log("⚡ BotManager khởi động ở chế độ không config")

    def _verify_api_connection(self):
        balance = get_balance(self.api_key, self.api_secret)
        if balance is None:
            self.log("❌ LỖI: Không thể kết nối Binance API.")
        else:
            self.log(f"✅ Kết nối Binance thành công! Số dư:
{balance:.2f} USDT")

    def log(self, message):
        logger.info(f"[SYSTEM] {message}")
        if self.telegram_bot_token and self.telegram_chat_id:
            send_telegram(f"<b>SYSTEM</b>: {message}",
                           bot_token=self.telegram_bot_token,
                           default_chat_id=self.telegram_chat_id)

    def send_main_menu(self, chat_id):
        welcome = "🤖 <b>BOT GIAO DỊCH FUTURES THÔNG MINH</b>\n\n📊
<b>HỆ THỐNG ĐA CHIẾN LƯỢC + SMART EXIT + BOT ĐỘNG</b>"
        send_telegram(welcome, chat_id, create_main_menu(),
                       bot_token=self.telegram_bot_token,
                       default_chat_id=self.telegram_chat_id)

    def _is_in_cooldown(self, strategy_type, config_key):
        """Kiểm tra xem chiến lược có đang trong thời gian chờ
không"""
        if strategy_type not in self.strategy_cooldowns:
            return False

        last_cooldown_time =
self.strategy_cooldowns[strategy_type].get(config_key)
        if last_cooldown_time is None:
            return False

```

```


current_time = time.time()
if current_time - last_cooldown_time < self.cooldown_period:
    return True


# Hết cooldown, xóa khỏi danh sách
if current_time - last_cooldown_time >= self.cooldown_period:
    del self.strategy_cooldowns[strategy_type][config_key]
return False

def _auto_scan_loop(self):
    """VÒNG LẶP TỰ ĐỘNG QUÉT COIN VỚI COOLDOWN"""
    while self.running:
        try:
            current_time = time.time()

            removed_count = 0
            for bot_id in list(self.bots.keys()):
                bot = self.bots[bot_id]
                # Chỉ xóa bot tĩnh đã đóng lệnh (bot động tự xử lý
chuyển coin)
                if (hasattr(bot, 'should_be_removed') and
bot.should_be_removed and
not getattr(bot, 'dynamic_mode', False)):


                    # Thêm cooldown cho chiến lược TĨNH VỪA ĐÓNG
LỆNH
                    strategy_type = bot.strategy_name
                    config_key = getattr(bot, 'config_key', None)
                    if config_key and strategy_type in
self.strategy_cooldowns:

self.strategy_cooldowns[strategy_type][config_key] = current_time
                    self.log(f" Đã thêm cooldown cho
{strategy_type} - {config_key}")

                    self.log(f" Tự động xóa bot {bot_id} (đã
đóng lệnh)")

                    self.stop_bot(bot_id)
                    removed_count += 1
                    time.sleep(0.5)

            if (removed_count > 0 or
current_time - self.last_auto_scan >
self.auto_scan_interval):

                if removed_count > 0:
                    self.log(f" Đã xóa {removed_count} bot tĩnh,
đợi 10s trước khi quét coin mới")

```



```

        # Chỉ tạo bot nếu chưa có bot cho coin này
        if bot_id not in self.bots and added_count <
(2 - current_bots_count):
            success = self._create_auto_bot(symbol,
strategy_type, strategy_config)
            if success:
                added_count += 1
                self.log(f"✅ Đã thêm {symbol} cho
{strategy_type} (Config: {strategy_key})")
                time.sleep(1) # Tránh rate limit

            if added_count > 0:
                self.log(f"🎯 {strategy_type}: đã thêm
{added_count} bot mới cho config {strategy_key}")
            else:
                self.log(f"⚠️ {strategy_type}: không tìm thấy
coin mới phù hợp cho config {strategy_key}")
            else:
                self.log(f"✅ {strategy_type} (Config:
{strategy_key}): đã đủ 2 bot, không tìm thêm")

        except Exception as e:
            self.log(f"❌ Lỗi quét {strategy_type}: {str(e)}")

    def _find_qualified_symbols(self, strategy_type, leverage, config,
strategy_key):
        """Tìm coin phù hợp cho chiến lược"""
        try:
            threshold = config.get('threshold', 30)
            volatility = config.get('volatility', 3)
            grid_levels = config.get('grid_levels', 5)

            # Sử dụng hàm get_qualified_symbols đã tích hợp
            qualified_symbols = get_qualified_symbols(
                self.api_key, self.api_secret, strategy_type,
leverage,
                threshold, volatility, grid_levels,
                max_candidates=20,
                final_limit=2, # Chỉ cần 2 coin để lấp đầy bot còn
thiếu
                strategy_key=strategy_key
            )

            return qualified_symbols

        except Exception as e:
            self.log(f"❌ Lỗi tìm coin: {str(e)}")
            return []

```



```

def _create_auto_bot(self, symbol, strategy_type, config):
    """Tạo bot tự động (dành cho auto_strategies)"""
    try:
        leverage = config['leverage']
        percent = config['percent']
        tp = config['tp']
        sl = config['sl']
        strategy_key = config['strategy_key']
        smart_exit_config = config.get('smart_exit_config', {})
        dynamic_mode = config.get('dynamic_mode', False)

        bot_class = {
            "Reverse 24h": Reverse_24h_Bot,
            "Scalping": Scalping_Bot,
            "Safe Grid": Safe_Grid_Bot,
            "Trend Following": Trend_Following_Bot,
            "Smart Dynamic": SmartDynamicBot
        }.get(strategy_type)

        if not bot_class:
            return False

        # Cần kiểm tra bot đã được đăng ký chưa
        if CoinManager().is_coin_managed(symbol):
            self.log(f"⚠️ {symbol} đã được quản lý, bỏ qua")
            return False

        # Tạo bot với tham số đặc biệt nếu có
        if strategy_type == "Reverse 24h":
            threshold = config.get('threshold', 30)
            bot = bot_class(symbol, leverage, percent, tp, sl,
self.ws_manager,
                                self.api_key, self.api_secret,
self.telegram_bot_token,
                                self.telegram_chat_id, threshold,
strategy_key, smart_exit_config, dynamic_mode)
        elif strategy_type == "Safe Grid":
            grid_levels = config.get('grid_levels', 5)
            bot = bot_class(symbol, leverage, percent, tp, sl,
self.ws_manager,
                                self.api_key, self.api_secret,
self.telegram_bot_token,
                                self.telegram_chat_id, grid_levels,
strategy_key, smart_exit_config, dynamic_mode)
        else:
            # Bot Dynamic/Static thông thường
            bot = bot_class(symbol, leverage, percent, tp, sl,

```

```

self.ws_manager,
                                self.api_key, self.api_secret,
self.telegram_bot_token,
                                self.telegram_chat_id, strategy_key,
smart_exit_config, dynamic_mode)

    bot_id = f"{symbol}_{strategy_key}"
    self.bots[bot_id] = bot
    return True

except Exception as e:
    self.log(f"❌ Lỗi tạo bot {symbol}: {str(e)}")
    return False

def add_bot(self, symbol, lev, percent, tp, sl, strategy_type,
**kwargs):
    """
    THÊM BOT MỚI - PHIÊN BẢN TỐI ƯU
    """
    try:
        if sl == 0: sl = None
        if not self.api_key or not self.api_secret:
            self.log(f"❌ Chưa thiết lập API Key trong BotManager")
            return False

        test_balance = get_balance(self.api_key, self.api_secret)
        if test_balance is None:
            self.log(f"❌ LỖI: Không thể kết nối Binance")
            return False

        smart_exit_config = kwargs.get('smart_exit_config', {})
        dynamic_mode = kwargs.get('dynamic_mode', False)
        threshold = kwargs.get('threshold')
        volatility = kwargs.get('volatility')
        grid_levels = kwargs.get('grid_levels')

        bot_created = False

        # 🔄 BOT ĐỘNG (Bao gồm Smart Dynamic và các chiến lược
        khác ở chế độ động)
        if dynamic_mode:
            if strategy_type == "Smart Dynamic":
                bot_created = self._create_smart_dynamic_bot(
                    lev, percent, tp, sl, smart_exit_config,
dynamic_mode
                )
            else:
                bot_created = self._create_dynamic_bot(

```

```

        strategy_type, lev, percent, tp, sl,
        smart_exit_config, threshold, volatility,
grid_levels
    )

    # 🤖 BOT TĨNH TRUYỀN THÔNG
    else:
        bot_created = self._create_static_bot(
            symbol, strategy_type, lev, percent, tp, sl,
smart_exit_config
        )

        return bot_created

    except Exception as e:
        self.log(f"❌ Lỗi nghiêm trọng trong add_bot: {str(e)}")
        import traceback
        self.log(f"🔍 Chi tiết lỗi: {traceback.format_exc()}")
        return False

    def _create_smart_dynamic_bot(self, lev, percent, tp, sl,
smart_exit_config, dynamic_mode):
        """TẠO BOT ĐỘNG THÔNG MINH"""
        try:
            strategy_key = f"SmartDynamic_{lev}_{percent}_{tp}_{sl}"

            if self._is_in_cooldown("Smart Dynamic", strategy_key):
                self.log(f"🕒 Smart Dynamic (Config: {strategy_key}):
đang trong cooldown")
                return False

            # Lưu cấu hình auto strategy
            self.auto_strategies[strategy_key] = {
                'strategy_type': "Smart Dynamic",
                'leverage': lev,
                'percent': percent,
                'tp': tp,
                'sl': sl,
                'strategy_key': strategy_key,
                'smart_exit_config': smart_exit_config,
                'dynamic_mode': True
            }

            qualified_symbols = self._find_qualified_symbols(
                "Smart Dynamic", lev,
self.auto_strategies[strategy_key], strategy_key
            )

```

```

        success_count = 0
        for symbol in qualified_symbols:
            bot_id = f"{symbol}_{strategy_key}"
            if bot_id not in self.bots:
                success = self._create_auto_bot(symbol, "Smart
Dynamic", self.auto_strategies[strategy_key])
                if success:
                    success_count += 1
                    time.sleep(0.5)

        if success_count > 0:
            success_msg = self._format_success_message(
                "Smart Dynamic", lev, percent, tp, sl,
                qualified_symbols[:success_count], strategy_key
            )
            self.log(success_msg)
            return True
        else:
            self.log("⚠ Smart Dynamic: chưa tìm thấy coin phù
hợp")

            return False

    except Exception as e:
        self.log(f"❌ Lỗi tạo Smart Dynamic bot: {str(e)}")
        return False

    def _create_dynamic_bot(self, strategy_type, lev, percent, tp, sl,
smart_exit_config, threshold, volatility, grid_levels):
        """TẠO BOT ĐỘNG CHO CÁC CHIẾN LƯỢC KHÁC"""
        try:
            # Tạo strategy key duy nhất
            strategy_key =
f"{strategy_type}_{lev}_{percent}_{tp}_{sl}"
            if strategy_type == "Reverse 24h":
                strategy_key += f"_th{threshold or 30}"
            elif strategy_type == "Scalping":
                strategy_key += f"_vol{volatility or 3}"
            elif strategy_type == "Safe Grid":
                strategy_key += f"_grid{grid_levels or 5}"

            if self._is_in_cooldown(strategy_type, strategy_key):
                self.log(f"🕒 {strategy_type} (Config:
{strategy_key}): đang trong cooldown")
                return False

            # Lưu cấu hình
            config = {
                'strategy_type': strategy_type, 'leverage': lev,

```

```

'percent': percent, 'tp': tp, 'sl': sl,
        'strategy_key': strategy_key, 'smart_exit_config':
smart_exit_config, 'dynamic_mode': True
    }
    if threshold: config['threshold'] = threshold
    if volatility: config['volatility'] = volatility
    if grid_levels: config['grid_levels'] = grid_levels

    self.auto_strategies[strategy_key] = config

    qualified_symbols = self._find_qualified_symbols(
        strategy_type, lev, config, strategy_key
    )

    success_count = 0
    for symbol in qualified_symbols:
        bot_id = f"{symbol}_{strategy_key}"
        if bot_id not in self.bots:
            success = self._create_auto_bot(symbol,
strategy_type, config)
            if success:
                success_count += 1
                time.sleep(0.5)

    if success_count > 0:
        success_msg =
self._format_success_message(strategy_type, lev, percent, tp, sl,
qualified_symbols[:success_count], strategy_key,
                                                                    threshold,
volatility, grid_levels)
        self.log(success_msg)
        return True
    else:
        self.log(f"⚠️ {strategy_type}: chưa tìm thấy coin phù
hợp")
        return False

except Exception as e:
    self.log(f"❌ Lỗi tạo {strategy_type} bot: {str(e)}")
    return False

def _create_static_bot(self, symbol, strategy_type, lev, percent,
tp, sl, smart_exit_config):
    """TẠO BOT TĨNH TRUYỀN THỐNG"""
    try:
        symbol = symbol.upper()
        bot_id = f"{symbol}_{strategy_type}_static"

```

```

if bot_id in self.bots:
    self.log(f"⚠ Đã có bot {strategy_type} cho {symbol}")
    return False

bot_class = {
    "RSI/EMA Recursive": RSI_EMA_Bot,
    "EMA Crossover": EMA_Crossover_Bot,
    "Reverse 24h": Reverse_24h_Bot,
    "Trend Following": Trend_Following_Bot,
    "Scalping": Scalping_Bot,
    "Safe Grid": Safe_Grid_Bot
}.get(strategy_type)

if not bot_class:
    self.log(f"❌ Chiến lược {strategy_type} không được hỗ trợ")
    return False

# Kiểm tra coin đã được quản lý chưa
if CoinManager().is_coin_managed(symbol):
    self.log(f"⚠ {symbol} đã được quản lý bởi bot khác")
    return False

# Tạo bot với tham số phù hợp
if strategy_type == "Reverse 24h":
    bot = bot_class(symbol, lev, percent, tp, sl,
self.ws_manager,
                                self.api_key, self.api_secret,
self.telegram_bot_token,
                                self.telegram_chat_id, threshold=30,
config_key=bot_id, smart_exit_config=smart_exit_config,
dynamic_mode=False)
    elif strategy_type == "Safe Grid":
        bot = bot_class(symbol, lev, percent, tp, sl,
self.ws_manager,
                                self.api_key, self.api_secret,
self.telegram_bot_token,
                                self.telegram_chat_id, grid_levels=5,
config_key=bot_id, smart_exit_config=smart_exit_config,
dynamic_mode=False)
    else:
        bot = bot_class(symbol, lev, percent, tp, sl,
self.ws_manager,
                                self.api_key, self.api_secret,
self.telegram_bot_token,
                                self.telegram_chat_id,
config_key=bot_id, smart_exit_config=smart_exit_config,

```

```

dynamic_mode=False)

        self.bots[bot_id] = bot
        self.log(f"✅ Đã thêm bot {strategy_type}: {symbol} | DB:
{lev}x | Vốn: {percent}% | TP/SL: {tp}%/{sl}%")
        return True

    except Exception as e:
        self.log(f"❌ Lỗi tạo bot tính {symbol}: {str(e)}")
        return False

    def _format_success_message(self, strategy_type, lev, percent, tp,
sl, symbols, strategy_key, threshold=None, volatility=None,
grid_levels=None):
        """ĐỊNH DẠNG THÔNG BÁO THÀNH CÔNG"""
        message = (
            f"✅ <b>ĐÃ TẠO {len(symbols)} BOT {strategy_type}</b>\n\n"
            f"🎯 Chi ế n l ư ợ c: {strategy_type}\n"
            f"💰 Đòn b ấ y: {lev}x\n"
            f"📊 % S ố d ư : {percent}%\n"
            f"🎯 TP: {tp}%\n"
            f"🛡 SL: {sl}%\n"
        )

        if threshold: message += f"📈 Ngưỡng: {threshold}%\n"
        if volatility: message += f"⚡ Bi ế n đ ộ n g: {volatility}%\n"
        if grid_levels: message += f"🛡 S ố l ệ n h: {grid_levels}\n"

        message += f"📁 Coin: {' , '.join(symbols)}\n\n"
        message += f"🔑 <b>Config Key:</b> {strategy_key}\n"
        message += f"🔄 <i>Bot sẽ tự động tìm coin mới sau khi đ ố n g
l ệ n h</i>"

        return message

    def stop_bot(self, bot_id):
        bot = self.bots.get(bot_id)
        if bot:
            bot.stop()
            self.log(f"🛑 Đã d ừ n g bot {bot_id}")
            # X ố a k h ỏ i auto_strategies n ế u là bot đ ộ n g
            config_key = getattr(bot, 'config_key', None)
            if config_key and config_key in self.auto_strategies:
                # X ố a n ế u k h ỏ n g c ò n bot n ào d ừ n g c ò n g f i g _ k e y n à y
                if CoinManager().count_bots_by_config(config_key) <=
1:

                    del self.auto_strategies[config_key]
                    self.log(f"🗑 Đã x ố a c ấ u h ì n h t ự đ ộ n g

```

```

{config_key}")

        del self.bots[bot_id]
        return True
    return False

def stop_all(self):
    self.log("🛑 Đang dừng tất cả bot...")
    for bot_id in list(self.bots.keys()):
        self.stop_bot(bot_id)
    self.ws_manager.stop()
    self.running = False
    self.log("🛑 Hệ thống đã dừng")

def _telegram_listener(self):
    # ... (Nội dung hàm lắng nghe Telegram)
    last_update_id = 0

    while self.running and self.telegram_bot_token:
        try:
            url =
f"https://api.telegram.org/bot{self.telegram_bot_token}/getUpdates?off
set={last_update_id+1}&timeout=30"
            response = requests.get(url, timeout=35)

            if response.status_code == 200:
                data = response.json()
                if data.get('ok'):
                    for update in data['result']:
                        update_id = update['update_id']
                        message = update.get('message', {})
                        chat_id = str(message.get('chat',
{})).get('id'))

                        text = message.get('text', '').strip()

                        if chat_id != self.telegram_chat_id:
                            continue

                        if update_id > last_update_id:
                            last_update_id = update_id

                        self._handle_telegram_message(chat_id,
text)

                    elif response.status_code == 409:
                        logger.error("Lỗi xung đột Telegram")
                        time.sleep(60)
                    else:
                        time.sleep(10)

```



```

except Exception as e:
    logger.error(f"Lỗi Telegram listener: {str(e)}")
    time.sleep(10)

def _handle_telegram_message(self, chat_id, text):
    user_state = self.user_states.get(chat_id, {})
    current_step = user_state.get('step')

    # Logic xử lý tin nhắn Telegram (giữ nguyên logic từ v16.py)
    # ... (Các khối if/elif cho từng bước tạo bot và lệnh chính)

    # XỬ LÝ CÁC BƯỚC TẠO BOT THEO THỨ TỰ
    if current_step == 'waiting_bot_mode':
        if text == '✗ Hủy bỏ':
            self.user_states[chat_id] = {}
            send_telegram("✗ Đã hủy thêm bot", chat_id,
create_main_menu(), self.telegram_bot_token, self.telegram_chat_id)
        elif text in ["🤖 Bot Tĩnh - Coin cụ thể", "🔄 Bot Động -
Tự tìm coin"]:
            if text == "🤖 Bot Tĩnh - Coin cụ thể":
                user_state['dynamic_mode'] = False
            else:
                user_state['dynamic_mode'] = True
            user_state['step'] = 'waiting_strategy'
            mode_text = "TĨNH" if not user_state['dynamic_mode']
        else "ĐỘNG"

            send_telegram(
                f"🎯 <b>ĐÃ CHỌN: BOT {mode_text}</b>\n\n"
                "Chọn chiến lược:",
                chat_id, create_strategy_keyboard(),
                self.telegram_bot_token, self.telegram_chat_id
            )

    elif current_step == 'waiting_strategy':
        if text == '✗ Hủy bỏ':
            self.user_states[chat_id] = {}
            send_telegram("✗ Đã hủy thêm bot", chat_id,
create_main_menu(), self.telegram_bot_token, self.telegram_chat_id)
        elif text in ["🤖 RSI/EMA Recursive", "📊 EMA Crossover",
"🎯 Reverse 24h", "📈 Trend Following", "⚡ Scalping", "🛡️ Safe Grid",
"🔄 Bot Động Thông Minh"]:
            strategy_map = {
                "🤖 RSI/EMA Recursive": "RSI/EMA Recursive", "📊
EMA Crossover": "EMA Crossover", "🎯 Reverse 24h": "Reverse 24h",
                "📈 Trend Following": "Trend Following", "⚡
Scalping": "Scalping", "🛡️ Safe Grid": "Safe Grid", "🔄 Bot Động Thông
Minh": "Smart Dynamic"

```

```

    }
    strategy = strategy_map[text]
    user_state['strategy'] = strategy
    if strategy == "Smart Dynamic":
user_state['dynamic_mode'] = True
        user_state['step'] = 'waiting_exit_strategy'

        strategy_descriptions = {
            "RSI/EMA Recursive": "Phân tích RSI + EMA đệ quy",
"EMA Crossover": "Giao cắt EMA nhanh/chậm",
            "Reverse 24h": "Đảo chiều biến động 24h", "Trend
Following": "Theo xu hướng giá",
            "Scalping": "Giao dịch tốc độ cao", "Safe Grid":
"Grid an toàn",
            "Smart Dynamic": "Bot động thông minh đa chi ế n
lược"
        }
        description = strategy_descriptions.get(strategy, "")
        mode_text = "ĐỘNG" if user_state.get('dynamic_mode')
else "TĨNH"

        send_telegram(
            f"🎯 <b>ĐÃ CHỌN: {strategy}</b>\n🤖 <b>Ch ế độ:
{mode_text}</b>\n\n{description}\n\nChọn chi ế n lược thoát lệnh:",
            chat_id, create_exit_strategy_keyboard(),
            self.telegram_bot_token, self.telegram_chat_id
        )

        elif current_step == 'waiting_exit_strategy':
            if text == '❌ Hủy bỏ':
                self.user_states[chat_id] = {}
                send_telegram("❌ Đã hủy thêm bot", chat_id,
create_main_menu(), self.telegram_bot_token, self.telegram_chat_id)
            elif text in ["🔄 Thoát lệnh thông minh", "⚡ Thoát lệnh
cơ bản", "🎯 Chỉ TP/SL cố định"]:
                if text == "🔄 Thoát lệnh thông minh":
                    user_state['exit_strategy'] = 'smart_full'
                    user_state['step'] = 'waiting_smart_config'
                    send_telegram("Chọn cấu hình Smart Exit:",
chat_id, create_smart_exit_config_keyboard(), self.telegram_bot_token,
self.telegram_chat_id)
                elif text == "⚡ Thoát lệnh cơ bản":
                    user_state['exit_strategy'] = 'smart_basic'
                    user_state['smart_exit_config'] =
{'enable_trailing': True, 'enable_time_exit': True,
'enable_support_resistance': False, 'trailing_activation': 30,
'trailing_distance': 15, 'max_hold_time': 6}
                    self._continue_bot_creation(chat_id, user_state)

```

```

        else:
            user_state['exit_strategy'] = 'traditional'
            user_state['smart_exit_config'] =
{'enable_trailing': False, 'enable_time_exit': False,
'enable_support_resistance': False}
            self._continue_bot_creation(chat_id, user_state)

    elif current_step == 'waiting_smart_config':
        if text == '✗ Hủy bỏ':
            self.user_states[chat_id] = {}
            send_telegram("✗ Đã hủy thêm bot", chat_id,
create_main_menu(), self.telegram_bot_token, self.telegram_chat_id)
        else:
            smart_config = {}
            if text == "Trailing: 30/15":
                smart_config = {'enable_trailing': True,
'enable_time_exit': True, 'enable_support_resistance': True,
'trailing_activation': 30, 'trailing_distance': 15, 'max_hold_time':
4}

                elif text == "Trailing: 50/20":
                    smart_config = {'enable_trailing': True,
'enable_time_exit': True, 'enable_support_resistance': True,
'trailing_activation': 50, 'trailing_distance': 20, 'max_hold_time':
6}

                    elif text == "Time Exit: 4h":
                        smart_config = {'enable_trailing': True,
'enable_time_exit': True, 'enable_support_resistance': True,
'trailing_activation': 25, 'trailing_distance': 12, 'max_hold_time':
4}

                        elif text == "Time Exit: 8h":
                            smart_config = {'enable_trailing': True,
'enable_time_exit': True, 'enable_support_resistance': True,
'trailing_activation': 40, 'trailing_distance': 18, 'max_hold_time':
8}

                            elif text == "Kết hợp Full":
                                smart_config = {'enable_trailing': True,
'enable_time_exit': True, 'enable_support_resistance': True,
'trailing_activation': 35, 'trailing_distance': 15, 'max_hold_time':
6}

                                elif text == "Cơ bản":
                                    smart_config = {'enable_trailing': True,
'enable_time_exit': True, 'enable_support_resistance': False,
'trailing_activation': 30, 'trailing_distance': 15, 'max_hold_time':
6}

                                    user_state['smart_exit_config'] = smart_config
                                    self._continue_bot_creation(chat_id, user_state)

```

```

# XỬ LÝ CÁC BƯỚC THAM SỐ ĐẶC BIỆT
elif current_step == 'waiting_threshold':
    if text == '✗ Hủy bỏ': self.user_states[chat_id] = {};
    send_telegram("✗ Đã hủy thêm bot", chat_id, create_main_menu(),
    self.telegram_bot_token, self.telegram_chat_id); return
    try:
        threshold = float(text)
        if threshold <= 0: send_telegram("⚠ Ngưỡng phải lớn
hơn 0.", chat_id, create_threshold_keyboard(),
self.telegram_bot_token, self.telegram_chat_id); return
        user_state['threshold'] = threshold
        user_state['step'] = 'waiting_leverage'
        send_telegram(f"✅ Ngưỡng biến động:
{threshold}%\n\nChọn đòn bẩy:", chat_id, create_leverage_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)
    except ValueError: send_telegram("⚠ Vui lòng nhập số hợp
lệ cho ngưỡng:", chat_id, create_threshold_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)

elif current_step == 'waiting_volatility':
    if text == '✗ Hủy bỏ': self.user_states[chat_id] = {};
    send_telegram("✗ Đã hủy thêm bot", chat_id, create_main_menu(),
self.telegram_bot_token, self.telegram_chat_id); return
    try:
        volatility = float(text)
        if volatility <= 0: send_telegram("⚠ Biến động phải
lớn hơn 0.", chat_id, create_volatility_keyboard(),
self.telegram_bot_token, self.telegram_chat_id); return
        user_state['volatility'] = volatility
        user_state['step'] = 'waiting_leverage'
        send_telegram(f"⚡ Biến động tối thiểu:
{volatility}%\n\nChọn đòn bẩy:", chat_id, create_leverage_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)
    except ValueError: send_telegram("⚠ Vui lòng nhập số hợp
lệ cho biến động:", chat_id, create_volatility_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)

elif current_step == 'waiting_grid_levels':
    if text == '✗ Hủy bỏ': self.user_states[chat_id] = {};
    send_telegram("✗ Đã hủy thêm bot", chat_id, create_main_menu(),
self.telegram_bot_token, self.telegram_chat_id); return
    try:
        grid_levels = int(text)
        if grid_levels <= 0: send_telegram("⚠ Số lệnh grid
phải lớn hơn 0.", chat_id, create_grid_levels_keyboard(),
self.telegram_bot_token, self.telegram_chat_id); return
        user_state['grid_levels'] = grid_levels
        user_state['step'] = 'waiting_leverage'

```

```

        send_telegram(f"🛡️ Số lệnh grid: {grid_levels}\n\nChọn
đòn bẩy:", chat_id, create_leverage_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)
        except ValueError: send_telegram("⚠️ Vui lòng nhập số hợp
lệ cho số lệnh grid:", chat_id, create_grid_levels_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)



# XỬ LÝ CÁC BƯỚC CƠ BẢN
elif current_step == 'waiting_symbol':
    if text == '❌ Hủy bỏ': self.user_states[chat_id] = {};
send_telegram("❌ Đã hủy thêm bot", chat_id, create_main_menu(),
self.telegram_bot_token, self.telegram_chat_id); return
    user_state['symbol'] = text
    user_state['step'] = 'waiting_leverage'
    send_telegram(f"🔗 Coin: {text}\n\nChọn đòn bẩy:",
chat_id, create_leverage_keyboard(), self.telegram_bot_token,
self.telegram_chat_id)


elif current_step == 'waiting_leverage':
    if text == '❌ Hủy bỏ': self.user_states[chat_id] = {};
send_telegram("❌ Đã hủy thêm bot", chat_id, create_main_menu(),
self.telegram_bot_token, self.telegram_chat_id); return
    lev_text = text[:-1] if text.endswith('x') else text
    try:
        leverage = int(lev_text)
        if leverage <= 0 or leverage > 100: send_telegram("⚠️
Đòn bẩy phải từ 1 đến 100.", chat_id, create_leverage_keyboard(),
self.telegram_bot_token, self.telegram_chat_id); return
        user_state['leverage'] = leverage
        user_state['step'] = 'waiting_percent'
        balance = get_balance(self.api_key, self.api_secret)
        balance_info = f"\n💰 Số dư hiện có: {balance:.2f}
USDT" if balance else ""
        send_telegram(f"💰 Đòn bẩy:
{leverage}x{balance_info}\n\nChọn % số dư cho mỗi lệnh:", chat_id,
create_percent_keyboard(), self.telegram_bot_token,
self.telegram_chat_id)
        except ValueError: send_telegram("⚠️ Vui lòng nhập số hợp
lệ cho đòn bẩy:", chat_id, create_leverage_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)

elif current_step == 'waiting_percent':
    if text == '❌ Hủy bỏ': self.user_states[chat_id] = {};
send_telegram("❌ Đã hủy thêm bot", chat_id, create_main_menu(),
self.telegram_bot_token, self.telegram_chat_id); return
    try:
        percent = float(text)
        if percent <= 0 or percent > 100: send_telegram("⚠️ %

```

```

số dư phải từ 0.1 đến 100.", chat_id, create_percent_keyboard(),
self.telegram_bot_token, self.telegram_chat_id); return
    user_state['percent'] = percent
    user_state['step'] = 'waiting_tp'
    balance = get_balance(self.api_key, self.api_secret)
    actual_amount = balance * (percent / 100) if balance
else 0
    send_telegram(f" % Số dư: {percent}%\n Số tiền mỗi
lệnh: ~{actual_amount:.2f} USDT\n\nChọn Take Profit (%):", chat_id,
create_tp_keyboard(), self.telegram_bot_token, self.telegram_chat_id)
    except ValueError: send_telegram("⚠️ Vui lòng nhập số hợp
lệ cho % số dư:", chat_id, create_percent_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)

    elif current_step == 'waiting_tp':
        if text == '❌ Hủy bỏ': self.user_states[chat_id] = {};
send_telegram("❌ Đã hủy thêm bot", chat_id, create_main_menu(),
self.telegram_bot_token, self.telegram_chat_id); return
        try:
            tp = float(text)
            if tp <= 0: send_telegram("⚠️ Take Profit phải lớn hơn
0.", chat_id, create_tp_keyboard(), self.telegram_bot_token,
self.telegram_chat_id); return
            user_state['tp'] = tp
            user_state['step'] = 'waiting_sl'
            send_telegram(f" Take Profit: {tp}%\n\nChọn Stop
Loss (%):", chat_id, create_sl_keyboard(), self.telegram_bot_token,
self.telegram_chat_id)
            except ValueError: send_telegram("⚠️ Vui lòng nhập số hợp
lệ cho Take Profit:", chat_id, create_tp_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)

    elif current_step == 'waiting_sl':
        if text == '❌ Hủy bỏ': self.user_states[chat_id] = {};
send_telegram("❌ Đã hủy thêm bot", chat_id, create_main_menu(),
self.telegram_bot_token, self.telegram_chat_id); return
        try:
            sl = float(text)
            if sl < 0: send_telegram("⚠️ Stop Loss phải lớn hơn
hoặc bằng 0.", chat_id, create_sl_keyboard(), self.telegram_bot_token,
self.telegram_chat_id); return
            user_state['sl'] = sl

        # THỰC HIỆN TẠO BOT
        strategy, dynamic_mode, leverage, percent, tp, sl,
symbol = (
            user_state.get('strategy'),
user_state.get('dynamic_mode', False), user_state.get('leverage'),

```

```

user_state.get('percent'),
            user_state.get('tp'), user_state.get('sl'),
user_state.get('symbol')
        )
        exit_strategy, smart_exit_config =
user_state.get('exit_strategy', 'traditional'),
user_state.get('smart_exit_config', {})
        threshold, volatility, grid_levels =
user_state.get('threshold'), user_state.get('volatility'),
user_state.get('grid_levels')

        success = self.add_bot(symbol, leverage, percent, tp,
sl, strategy, dynamic_mode=dynamic_mode,
smart_exit_config=smart_exit_config, threshold=threshold,
volatility=volatility, grid_levels=grid_levels)

        if success:
            mode_text = "ĐỘNG" if dynamic_mode else "TĨNH"
            success_msg = (
                f"✅ <b>ĐÃ TẠO BOT THÀNH CÔNG</b>\n\n🤖 Chi ế n
lược: {strategy}\n🔧 Chê độ: {mode_text}\n💰 Đòn bẩy: {leverage}x\n📊
% Số dư: {percent}%\n🎯 TP: {tp}%\n🛡️ SL: {sl}%"
            )
            if not dynamic_mode: success_msg += f"\n🔗 Coin:
{symbol}"
            if dynamic_mode: success_msg += f"\n\n🔄 <i>Bot sẽ
tự động tìm coin mới sau mỗi lệnh</i>"

            send_telegram(success_msg, chat_id,
create_main_menu(), self.telegram_bot_token, self.telegram_chat_id)
        else:
            send_telegram("❌ Có lỗi khi tạo bot. Vui lòng thử
lại.", chat_id, create_main_menu(), self.telegram_bot_token,
self.telegram_chat_id)

        self.user_states[chat_id] = {}

        except ValueError: send_telegram("⚠️ Vui lòng nhập số hợp
lệ cho Stop Loss:", chat_id, create_sl_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)

# XỬ LÝ CÁC LỆNH CHÍNH
elif text == "➕ Thêm Bot":
    self.user_states[chat_id] = {'step': 'waiting_bot_mode'}
    balance = get_balance(self.api_key, self.api_secret)
    if balance is None:
        send_telegram("❌ <b>LỖI KẾT NỐI BINANCE</b>\nVui lòng
kiểm tra API Key!", chat_id, self.telegram_bot_token,

```

```

self.telegram_chat_id)
    return
    send_telegram(f"🎯 <b>CHỌN CHẾ ĐỘ BOT</b>\n\n💰 Số dư hiện
có: <b>{balance:.2f}</b> USDT</b>\n\n🤖 <b>Bot Tĩnh:</b>\n• Giao dịch coin
CỔ ĐỊNH\n🔄 <b>Bot Động:</b>\n• TỰ ĐỘNG tìm coin tốt nhất\n• Tự chuyển
coin sau mỗi lệnh", chat_id, create_bot_mode_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)

    elif text == "📊 Danh sách Bot":
        if not self.bots:
            send_telegram("🤖 Không có bot nào đang chạy",
chat_id, self.telegram_bot_token, self.telegram_chat_id)
        else:
            message = "🤖 <b>DANH SÁCH BOT ĐANG CHẠY</b>\n\n"
            for bot_id, bot in self.bots.items():
                status = "🟢 Mở" if bot.status == "open" else "🟡
Chờ"
                mode = "🔄 Động" if getattr(bot, 'dynamic_mode',
False) else "🤖 Tĩnh"
                exit_type = "🔴 Thường"
                if hasattr(bot, 'smart_exit') and
bot.smart_exit.config.get('enable_trailing'): exit_type = "🟢 Thông
minh"
                message += f"💎 {bot_id} | {status} | {mode} |
{exit_type} | {bot.strategy_name}\n"
            send_telegram(message, chat_id,
self.telegram_bot_token, self.telegram_chat_id)

    elif text == "🛑 Dừng Bot":
        if not self.bots: send_telegram("🤖 Không có bot nào đang
chạy", chat_id, self.telegram_bot_token, self.telegram_chat_id)
        else:
            message = "🛑 <b>CHỌN BOT ĐỂ DỪNG</b>\n\n"
            keyboard, row = [], []
            for i, bot_id in enumerate(self.bots.keys()):
                message += f"💎 {bot_id}\n"; row.append({"text":
f"🛑 {bot_id}"})
                if len(row) == 2 or i == len(self.bots) - 1:
                    keyboard.append(row); row = []
            keyboard.append([{"text": "❌ Hủy bỏ"}])
            send_telegram(message, chat_id, {"keyboard": keyboard,
"resize_keyboard": True, "one_time_keyboard": True},
self.telegram_bot_token, self.telegram_chat_id)

    elif text.startswith("🛑 "):
        bot_id = text.replace("🛑 ", "").strip()
        if self.stop_bot(bot_id): send_telegram(f"🛑 Đã dừng bot
{bot_id}", chat_id, create_main_menu(), self.telegram_bot_token,

```



```

self.telegram_chat_id)
    else: send_telegram(f"⚠ Không tìm thấy bot {bot_id}",
chat_id, create_main_menu(), self.telegram_bot_token,
self.telegram_chat_id)

    elif text == "💰 Số dư":
        balance = get_balance(self.api_key, self.api_secret)
        if balance is None: send_telegram("❌ <b>LỖI KẾT NỐI
BINANCE</b>", chat_id, self.telegram_bot_token, self.telegram_chat_id)
        else: send_telegram(f"💰 <b>SỐ DƯ KHẢ DỤNG</b>:
{balance:.2f} USDT", chat_id, self.telegram_bot_token,
self.telegram_chat_id)

    elif text == "📈 Vị thế":
        positions = get_positions(api_key=self.api_key,
api_secret=self.api_secret)
        if not positions: send_telegram("🚫 Không có vị thế nào
đang mở", chat_id, self.telegram_bot_token, self.telegram_chat_id);
        return

        message = "📈 <b>VỊ THẾ ĐANG MỞ</b>\n\n"
        for pos in positions:
            position_amt, entry, pnl =
float(pos.get('positionAmt', 0)), float(pos.get('entryPrice', 0)),
float(pos.get('unRealizedProfit', 0))
            if position_amt != 0:
                side = "LONG" if position_amt > 0 else "SHORT"
                message += f"💎 {pos.get('symbol')} | {side}\n📊
Khối lượng: {abs(position_amt):.4f}\n👉 Giá vào: {entry:.4f}\n💰 PnL:
{pnl:.2f} USDT\n\n"
                send_telegram(message, chat_id, self.telegram_bot_token,
self.telegram_chat_id)

    elif text == "🎯 Chiến lược":
        strategy_info = ("🎯 <b>DANH SÁCH CHIẾN LƯỢC HOÀN
CHỈNH</b>\n\n🔄 <b>Bot Động Thông Minh</b>\n• Tự động tìm coin + Smart
Exit\n\n
        "🎯 <b>Reverse 24h</b> - HỖ TRỢ
ĐỘNG/TĨNH\n⚡ <b>Scalping</b> - HỖ TRỢ ĐỘNG/TĨNH\n🛡 <b>Safe Grid</b>
- HỖ TRỢ ĐỘNG/TĨNH\n📈 <b>Trend Following</b> - HỖ TRỢ ĐỘNG/TĨNH\n\n
        "🤖 <b>RSI/EMA Recursive</b> - TĨNH\n📊
<b>EMA Crossover</b> - TĨNH\n💡 <b>Smart Exit System</b>\n• 🔄
Trailing Stop\n• ⌚ Time Exit\n• 📊 Volume Exit\n• 🎯
Support/Resistance Exit")
        send_telegram(strategy_info, chat_id,
self.telegram_bot_token, self.telegram_chat_id)

    elif text == "⚙ Cấu hình":
        balance = get_balance(self.api_key, self.api_secret)

```

```

        api_status = "✅ Đã kết nối" if balance is not None else
"❌ Lỗi kết nối"
        dynamic_bots_count = sum(1 for bot in self.bots.values()
if getattr(bot, 'dynamic_mode', False))
        config_info = (f"⚙️ <b>CẤU HÌNH HỆ THỐNG THÔNG
MINH</b>\n\n🔑 Binance API: {api_status}\n🤖 Tổng số bot:
{len(self.bots)}\n🔄 Bot động: {dynamic_bots_count}\n🌐 WebSocket:
{len(self.ws_manager.connections)} kết nối\n🕒 Cooldown:
{self.cooldown_period//60} phút")
        send_telegram(config_info, chat_id,
self.telegram_bot_token, self.telegram_chat_id)

    elif text:
        self.send_main_menu(chat_id)

def _continue_bot_creation(self, chat_id, user_state):
    """Tiếp tục quy trình tạo bot sau khi chọn Smart Exit"""
    strategy = user_state.get('strategy')
    dynamic_mode = user_state.get('dynamic_mode', False)

    if dynamic_mode and strategy != "Smart Dynamic":
        # Yêu cầu tham số đặc biệt cho bot động
        if strategy == "Reverse 24h":
            user_state['step'] = 'waiting_threshold'
            send_telegram(f"🎯 <b>BOT ĐỘNG: {strategy}</b>\n\nChọn
ngưỡng biến động (%):", chat_id, create_threshold_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)
        elif strategy == "Scalping":
            user_state['step'] = 'waiting_volatility'
            send_telegram(f"🎯 <b>BOT ĐỘNG: {strategy}</b>\n\nChọn
biến động tối thiểu (%):", chat_id, create_volatility_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)
        elif strategy == "Safe Grid":
            user_state['step'] = 'waiting_grid_levels'
            send_telegram(f"🎯 <b>BOT ĐỘNG: {strategy}</b>\n\nChọn
số lệnh grid:", chat_id, create_grid_levels_keyboard(),
self.telegram_bot_token, self.telegram_chat_id)
        else:
            user_state['step'] = 'waiting_leverage'
            send_telegram(f"🎯 <b>BOT ĐỘNG: {strategy}</b>\n\nChọn
đòn bẩy:", chat_id, create_leverage_keyboard(strategy),
self.telegram_bot_token, self.telegram_chat_id)
    else:
        if not dynamic_mode:
            # Bot tĩnh
            user_state['step'] = 'waiting_symbol'
            send_telegram(f"🎯 <b>BOT TĨNH: {strategy}</b>\n\nChọn
cặp coin:", chat_id, create_symbols_keyboard(strategy),

```

```

self.telegram_bot_token, self.telegram_chat_id)
    else:
        # Bot động thông minh
        user_state['step'] = 'waiting_leverage'
        send_telegram(f"🎯 <b>BOT ĐỘNG THÔNG MINH</b>\n\nChọn
đòn bẩy:", chat_id, create_leverage_keyboard(strategy),
self.telegram_bot_token, self.telegram_chat_id)

# ===== KHỞI TẠO GLOBAL INSTANCES =====
coin_manager = CoinManager()

```