ChatGPT

GPT-4 Powered MetaTrader5 Trading Bots

Overview

This solution provides two Python trading bot scripts (single-asset and multi-asset) that integrate MetaTrader 5 (MT5) with OpenAI's GPT-4 API to automate trading decisions. The bots fetch real-time **5-minute (M5) price data** from MT5, compute key technical indicators (RSI, MACD, Bollinger Bands, moving averages), and feed this information into GPT-4 to receive trade recommendations. The AI's response includes a **trade action** (buy, sell, or hold), a **volume** (lot size), and suggested **stop-loss (SL)** and **take-profit (TP)** levels. The script then validates the signal and executes the trade via the MT5 Python API, logging all prompts and decisions for transparency and analysis. This approach leverages GPT-4's natural language processing to interpret market data and generate precise trading signals with entry and exit levels [1] [2].

Key Features:
- **Real-Time Data & Indicators:** Continuously retrieves M5 OHLC data for one or multiple assets from MT5 and calculates RSI, MACD (12,26,9), Bollinger Bands (20-period, 2σ), and moving averages (e.g. 50 and 200 period) in Python [3] [4].
- **AI-Based Decisions:** Sends a prompt containing recent price history and indicator values to OpenAI's API (GPT-4 model) and receives an **actionable signal** (BUY/SELL or HOLD) with recommended lot size, SL, and TP [1] [2].
- **Trade Execution:** Parses the AI response and, if a trade signal is given, submits an order via `MetaTrader5.order_send()` with the specified parameters (market order with SL/TP) [5]. It uses a small price **deviation** (slippage tolerance) and includes a unique trade comment and magic number for tracking.
- **Logging & Monitoring:** All prompts, AI responses, and trade results are logged to a file (or console) with timestamps for auditability. This includes whether trades were executed successfully or if any errors occurred.
- **Secure API Access:** The OpenAI API key is loaded from an environment variable (e.g. `OPENAI_API_KEY`) for security – **no hard-coded keys** in the script [6] [7]. This prevents accidental exposure of the secret key and follows best practices by treating it like a password.
- **Fault Tolerance & Safety:** The bot is designed with error handling to ensure stability. It checks for valid data and API responses, handles exceptions (like network issues or API errors) by safely retrying or skipping a cycle, and will not execute trades on unrecognized or unsafe instructions. The volume and SL/TP values are validated to be reasonable (e.g., non-zero, within instrument ranges) before sending orders. If the AI suggests "hold" or provides an incomplete response, the bot will log it and not trade.

Setup and Dependencies

Prerequisites:
- **MetaTrader 5 Terminal:** Install MetaTrader 5 on Windows 10+ and **log in to a broker account** (demo or live). Ensure the MT5 terminal is running and **Algo Trading is enabled** (if required for API trading).
- **Python Environment:** Install Python 3.7+ on Windows. It's recommended to create a virtual environment.
- **MT5 Python Package:** Install the official MetaTrader5 Python module (`pip install MetaTrader5`). This allows Python to connect to the running MT5 terminal and perform trade

operations.
- **OpenAI API Package:** Install OpenAI's Python SDK (`pip install openai`). This is used to call GPT-4.
- **Data and Indicator Libraries:** Install Pandas for data handling (`pip install pandas`). Optionally, install a TA library like **pandas_ta** for technical indicators (`pip install pandas_ta`), or we will compute indicators manually using Pandas/Numpy.
- **Logging and Env Vars:** We use Python's built-in `logging` for writing logs. For secure API key management, set an environment variable **OPENAI_API_KEY** to your OpenAI key. For example, in Windows Command Prompt: `set OPENAI_API_KEY=<your_api_key>`. You can also create a `.env` file and use `python-dotenv` to load it if preferred. The key should **never be hard-coded** in the script [6] [7].

Environment Variable Setup:
Ensure your OpenAI API key is stored in the environment. The script will retrieve it as shown below:

```
import os, openai
openai.api_key = os.getenv("OPENAI_API_KEY")
```

This line fetches the key from the environment and configures the OpenAI client [6]. If the environment variable is not set, the script will be unable to authenticate – you should handle this by exiting with an error message prompting the user to set the variable. Never commit your API key to code or logs [7].

Dependencies Installation:

```
pip install MetaTrader5 openai pandas pandas_ta python-dotenv
```

- *MetaTrader5:* Official MetaQuotes API for MT5 [8]
- *openai:* OpenAI Python client for GPT-3/4
- *pandas:* DataFrame library for managing OHLC data
- *pandas_ta: (Optional)* Technical analysis indicators library (for RSI, MACD, etc.) [4]
- *python-dotenv: (Optional)* to load environment variables from a .env file

How the Bots Work

Both scripts follow a similar flow, with the portfolio version simply looping over multiple symbols:

1. **Initialize MT5 Connection:** The script calls `mt5.initialize()` to connect Python to the running MetaTrader 5 terminal. It can also ensure the desired symbols are available in Market Watch (using `mt5.symbol_select()` if needed).
2. **Data Retrieval:** On each cycle (e.g., every 5 minutes), the bot fetches recent price data for the asset(s). We use `mt5.copy_rates_from_pos(symbol, mt5.TIMEFRAME_M5, start_pos, count)` or `copy_rates_from` to get the latest M5 bars into a Pandas DataFrame [8]. Typically, we retrieve a window of the last N bars (for instance, 100 or 300 bars) to compute indicators.
3. **Indicator Calculation:** Using the price DataFrame, the script computes technical indicators:
4. *RSI (14-period):* Measures momentum of price changes. (We compute gain/loss averages over 14 bars and derive RSI, or use `pandas_ta.rsi` for convenience.)

5. *MACD (12-26-9):* Computes fast and slow EMAs (12, 26) and their difference, plus the signal line (9-period EMA of the difference). The result includes MACD line and signal line (and optionally histogram).
6. *Bollinger Bands (20, 2σ):* Calculates the 20-period moving average and the standard deviation. The upper band = MA20 + 2*std, lower band = MA20 – 2*std. These bands indicate volatility and possible price extremes.
7. *Moving Averages:* Computes fast and slow simple moving averages (e.g., 50-bar and 200-bar SMA) to gauge trend direction or crossovers.

These values are prepared for the prompt. (The MT5 Python API itself does not provide technical indicators directly [3], so we compute them in Python, using libraries like **TA-Lib/pandas_ta** or custom code.) [3] [4]

4. **Prompt Creation:** The bot formats a text prompt for GPT-4 that includes the relevant market data. For example, the prompt may look like:

"Asset: EURUSD (5m timeframe)\nRecent Prices (OHLC): [1.0910, 1.0925, 1.0900, 1.0918] ... (last few bars)\nIndicators:\n- RSI(14): 65 (uptrend, near overbought)\n- MACD: 0.0005 (above signal by 0.0003)\n- Bollinger Bands: Upper 1.0950, Lower 1.0850 (price near upper band)\n- 50MA: 1.0905, 200MA: 1.0890 (short-term above long-term)\nQuestion: What is the recommended action (Buy, Sell, or Hold) for this asset? Provide volume (in lots), stop-loss and take-profit levels (pips or price) with brief reasoning."

The prompt is constructed to give GPT-4 a clear picture of recent market behavior and indicator status, then explicitly asks for a trading decision and specifics. We also **instruct the model to be concise and output in a structured format** (for easier parsing). For example, we might ask it to respond in JSON or a fixed format like: `{"action": "...", "volume": ..., "stop_loss": "...", "take_profit": "..."}`. This prompt engineering ensures the response contains the fields we need.

5. **Calling OpenAI API:** Using the OpenAI Python SDK, the script sends the prompt to GPT-4. We use the ChatCompletion endpoint with `model="gpt-4"`, including a system message to set the AI's role (e.g. "You are a trading assistant... respond with a single trade action.") and the user message as the prompt. We typically set `temperature=0.0` for a deterministic response, so the output format is consistent. The API call might look like:

```
response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[
        {"role": "system", "content": "You are a financial trading
assistant..."},
        {"role": "user", "content": prompt_text}
    ],
    temperature=0.0
)
decision_text = response['choices'][0]['message']['content']
```

The model's reply contains the recommended action, volume, and SL/TP. Using GPT-4 for this task allows complex analysis of the data in a flexible, natural language way [9] [2]. *(Note:* If GPT-4 access is limited, `gpt-3.5-turbo` could be used as an alternative model, possibly with slightly different prompt tuning.)

6. **Parsing the AI Response:** The script then interprets the AI's output. If we instructed a JSON format, we can `json.loads(decision_text)` to get a dictionary. If it's free-form text, we look for keywords

like “Buy/Sell/Hold”, numbers for volume, and SL/TP values. The parsing logic includes:
- Normalize the text to lower-case for scanning.
- Identify the **action** (buy/sell/hold). If the action is "hold" (or no trade), the bot will log the recommendation but **not execute any trade**.
- Extract the **volume** (e.g., a float or integer). The bot may cap this to a maximum allowed or adjust to account minimum (for instance, 0.01 lots for FX).
- Extract **stop-loss** and **take-profit**. These might be given as price levels or as pips. If the AI says “stop-loss 20 pips” and “take-profit 50 pips”, the script will convert those to price values based on the symbol’s pip size. For example, for EURUSD, 20 pips = 0.0020 in price. We use `symbol_info = mt5.symbol_info(symbol)` to get `point` (the smallest price increment) and `digits` (pricing precision) to handle this conversion. If values are already given as prices (e.g., 1.0880), we use them directly.
- Basic validation: ensure SL and TP are numeric and on the correct side of the current price (for a buy, SL < price < TP; for a sell, SL > price > TP, typically). If the model’s suggestion doesn’t make sense or is missing fields, the bot can either skip execution or apply defaults (e.g., no trade or use a preset SL/TP).
7. **Executing the Trade:** If a **Buy** or **Sell** signal is given and passes validation, the bot creates an order request dictionary for `mt5.order_send()`. For a market order on MT5, the request includes:

```
request = {
    "action": mt5.TRADE_ACTION_DEAL,
    "symbol": symbol,
    "volume": volume_lots,
    "type": mt5.ORDER_TYPE_BUY,  # or ORDER_TYPE_SELL
    "price": current_price,
# current market price (ask for buy, bid for sell)
    "sl": stop_loss_price,       # stop-loss price
    "tp": take_profit_price,     # take-profit price
    "deviation": 10,            # max slippage in points
    "magic": 123456,            # an arbitrary EA ID for tracking
    "comment": "GPT-4 signal",
    "type_time": mt5.ORDER_TIME_GTC,      # good till cancelled
    "type_filling": mt5.ORDER_FILLING_RETURN
}
result = mt5.order_send(request)
```

This will execute an immediate market order with the specified SL/TP attached [5]. The script then checks `result.retcode` to see if the trade was successful (MT5 returns `TRADE_RETCODE_DONE` if executed). If not successful, it logs the error (and can retrieve `mt5.last_error()` or details from `result` for debugging). On success, the bot logs the trade ID, and could also store that position to monitor or manage (e.g., for closing logic, though in this simple bot we rely on SL/TP to exit).
8. **Logging:** Every cycle, whether a trade was made or not, is logged. The log includes the timestamp, the prompt (or a summary of the data), the AI’s response, and the action taken. For example:

```
2025-08-03 16:00:00 | EURUSD | Prompt: {...} | GPT Response: {"action":
"BUY", "volume": 0.1, "stop_loss": "1.0900", "take_profit": "1.0950"} |
Executed BUY 0.1 lots @1.0920, SL=1.0900, TP=1.0950
```

Logs are invaluable for **monitoring bot performance and debugging**. They also help in prompt tuning: by reviewing cases where GPT's output was suboptimal, you can refine the prompt or handling logic.

1. **Loop Timing:** The bot repeats the above process continuously. You can run the loop with a sleep of ~5 minutes (300 seconds) to align with new M5 bars. Alternatively, synchronize to MT5's time: e.g., run at mm:00 and mm:05 etc. The portfolio bot will loop through each asset sequentially each cycle.

2. **Shutdown:** When stopping the bot (e.g., via keyboard interrupt or a stop condition), the script calls `mt5.shutdown()` to gracefully disconnect from the MT5 terminal.

Security & Safety Precautions

When deploying these bots, consider the following precautions:

- **API Key Security:** As emphasized, never hard-code your OpenAI API key in the script. Use environment variables or secure storage [7]. This prevents accidental leakage of the key (for instance, if you share your code). Always keep the key secret just like a password.
- **Cost Management:** Calling GPT-4 for every cycle incurs API costs. Monitor the frequency of calls and consider using `gpt-3.5-turbo` for more frequent or real-time strategies to reduce cost. You can also **batch requests** or limit tokens in the response to manage usage [10]. For example, restrict the prompt to essential data (last N bars or summary statistics rather than every price) to keep token count low.
- **Testing in Demo:** Always test the bots on a **demo account** or in strategy tester if possible before deploying on a live account. Verify that the logic works as expected, the AI outputs are sensible, and trades execute correctly. Monitor performance over a range of market conditions.
- **Risk Management:** Implement limits to ensure the AI's suggestions stay within your risk tolerance. For example, cap the volume (lot size) to a percentage of account equity or a fixed maximum. Ignore or override obviously risky outputs (if the AI ever suggested an unusually large trade or extremely tight/loose SL). You might also enforce a minimum stop-loss distance to avoid trades with SL too close to current price (which often get rejected by brokers).
- **Error Handling:** The scripts include try/except blocks around API calls and trade execution. If the OpenAI API call fails (network error, rate limit, etc.), the code should catch the exception and, for instance, log the error and skip that cycle (or retry after a delay), rather than crash. Similarly, handle `mt5.order_send` failures gracefully – log the `result.retcode` and do not retry the order repeatedly if it fails for conditions like market closed or no balance.
- **Prompt Safety:** Since the prompt includes live data, ensure that values are correctly formatted to avoid confusing the model. We also maintain a consistent prompt structure so GPT-4 knows what to output. It's wise to explicitly instruct the model to *only* output the decision and numbers, and not any overly verbose text. This simplifies parsing and reduces the chance of the model producing irrelevant content.
- **Model Constraints:** Keep in mind GPT-4 (as of 2025) does not have browsing or live data access – it relies purely on the prompt you provide. It's using the indicators and prices you feed it to make a judgment. The **quality of the prompt** will greatly affect the results. If you find the model's output not aligning with common-sense trading (e.g., it suggests a buy when all indicators are bearish), you may need to refine how the data is presented or add a rule-based sanity check. Consider prompt tuning as described below.
- **Logging & Monitoring:** Review the logs regularly. They will show if the AI is generally profitable or if it's making mistakes. If certain patterns of errors appear (e.g., always buying in a

downtrend), you can adjust either the prompt or add corrective logic (like a simple trend filter: e.g., if price under 200MA and falling, maybe ignore "buy" signals unless strong reason).

Prompt Tuning Tips

Getting optimal results from GPT-4 for trading decisions may require some experimentation:

- **Clarity:** Be very clear in the prompt about what you want. If you need a single-word action (Buy/Sell/Hold) plus numbers, ask for exactly that. E.g., *"Respond with one of: Buy, Sell, or Hold. Provide volume in lots (e.g., 0.10), a stop-loss and a take-profit. Use the latest data provided."* This reduces the chance the model returns a long explanation or misses a field.
- **Format Enforcement:** As mentioned, requesting a JSON or comma-separated format can simplify parsing. For example: *"Output in JSON like:* `{"action": "...", "volume": ..., "stop_loss": ..., "take_profit": ...}` *with numeric values."* GPT-4 is usually good at following this instruction [11]. If it occasionally deviates, you might use a regex to find the parts, or use GPT's functions or tools (if available) to validate format.
- **Include Context:** You might incorporate a brief strategy hint in the system message. For example, *"You are an expert financial analyst. You base decisions on trend, momentum, and volatility. Be cautious in choppy markets. If signals are mixed, recommend hold."* This can guide the AI's decision-making to be more aligned with what a human expert might do. But use this carefully – too much instruction can also confuse the model or make it too hesitant.
- **Indicator Interpretation:** Ensure the model knows what each indicator means. GPT-4 generally knows RSI, MACD, etc., but it might help to contextualize: e.g., *"RSI(14) = 80 (overbought >70), MACD just crossed above signal, Bollinger upper band at 1.2345..."* This way, the model gets not just raw numbers but their implication (overbought, bullish crossover, etc.). However, keep it concise to avoid hitting token limits.
- **Temperature and Max Tokens:** As noted, a low temperature (0 to 0.3) will make outputs more deterministic and likely to stick to the format. Use `max_tokens` large enough to get the full response, but you typically don't need more than, say, 50-100 tokens for a short decision output.
- **Test Prompts Offline:** It's useful to test your prompt on the OpenAI Playground or with a few static examples (maybe using historical data) to see how GPT responds. Tweak the wording until the output is consistently well-formatted and sensible, then embed that prompt structure into your bot.

Code Examples

Below are the two Python scripts: one for a single asset and one for a multi-asset portfolio. Both are heavily commented for clarity. **Before running**, make sure you have configured the environment variable for the API key and installed the required packages. Run these in a terminal or an IDE on Windows where MT5 is installed and running. (For clarity, the code uses a simple infinite loop; in practice you might integrate this into a scheduler or Windows service for continuous execution.)

Single-Asset Trading Bot (`mt5_gpt_single.py`)

This bot focuses on one symbol (defined by `SYMBOL`). It fetches the latest data for that symbol, gets a GPT-4 decision, and executes it. You can change `SYMBOL` to your asset of choice (e.g., `'EURUSD'`) and adjust the indicator parameters as needed.

```
import os, time, logging
import MetaTrader5 as mt5
```

```
import pandas as pd
# If using pandas_ta for indicators:
# import pandas_ta as ta

# === Configuration ===
SYMBOL = "EURUSD"            # trading symbol
TIMEFRAME = mt5.TIMEFRAME_M5 # 5-minute timeframe
BARS_COUNT = 300             # number of bars to retrieve for analysis
RSI_PERIOD = 14
FAST_MA = 12; SLOW_MA = 26; MACD_SIGNAL=9   # MACD parameters
BB_PERIOD = 20; BB_STDDEV = 2               # Bollinger Bands parameters
SMA_FAST = 50; SMA_SLOW = 200               # Moving averages for trend

# Logging setup
logging.basicConfig(filename="trading_bot.log",
                    level=logging.INFO,
                    format="%(asctime)s [%(levelname)s] %(message)s")
logging.info("Starting single-asset trading bot for %s", SYMBOL)

# Initialize MT5
if not mt5.initialize():
    logging.error("MT5 initialize() failed, error code=%s", mt5.last_error())
    raise SystemExit("MT5 initialization failed")

# Ensure the symbol is available
symbol_info = mt5.symbol_info(SYMBOL)
if symbol_info is None:
    logging.error("Symbol %s not found, exiting.", SYMBOL)
    mt5.shutdown()
    raise SystemExit(f"Symbol {SYMBOL} not found")

if not symbol_info.visible:
    mt5.symbol_select(SYMBOL, True)

# Load OpenAI API key from environment
import openai
openai.api_key = os.getenv("OPENAI_API_KEY")
if openai.api_key is None:
    logging.error("OPENAI_API_KEY not set. Please set the environment
variable.")
    mt5.shutdown()
    raise SystemExit("Missing OpenAI API key")

# Optional: you could set up openai Proxy or organization if needed
# openai.proxy = {...}

# Function to compute technical indicators on price DataFrame
def compute_indicators(df):
    # Ensure we have enough data
    if len(df) < max(RSI_PERIOD, FAST_MA, SLOW_MA, BB_PERIOD, SMA_SLOW):
        return None  # Not enough data to compute all indicators
```

```
    # Compute RSI
    delta = df['close'].diff()
    up = delta.clip(lower=0); down = -delta.clip(upper=0)
    # Calculate exponential moving average of ups and downs
    roll_up = up.ewm(span=RSI_PERIOD, adjust=False).mean()
    roll_down = down.ewm(span=RSI_PERIOD, adjust=False).mean()
    # Avoid division by zero
    RS = roll_up / (roll_down.replace(0, 1e-10))
    RSI = 100 - (100 / (1 + RS))
    rsi_current = RSI.iloc[-1]

    # Compute MACD (EMA12-EMA26 and signal EMA9)
    ema_fast = df['close'].ewm(span=FAST_MA, adjust=False).mean()
    ema_slow = df['close'].ewm(span=SLOW_MA, adjust=False).mean()
    macd_line = ema_fast - ema_slow
    signal_line = macd_line.ewm(span=MACD_SIGNAL, adjust=False).mean()
    macd_current = macd_line.iloc[-1]
    signal_current = signal_line.iloc[-1]

    # Compute Bollinger Bands (20, 2)
    ma = df['close'].rolling(BB_PERIOD).mean()
    sd = df['close'].rolling(BB_PERIOD).std()
    bb_upper = ma + BB_STDDEV * sd
    bb_lower = ma - BB_STDDEV * sd
    bb_upper_current = bb_upper.iloc[-1]
    bb_lower_current = bb_lower.iloc[-1]
    ma_current = ma.iloc[-1]  # middle band

    # Compute moving averages for trend
    sma_fast = df['close'].rolling(SMA_FAST).mean().iloc[-1]
    sma_slow = df['close'].rolling(SMA_SLOW).mean().iloc[-1] if len(df) >=
SMA_SLOW else None

    indicators = {
        "rsi": round(float(rsi_current), 2),
        "macd": round(float(macd_current), 6),
        "macd_signal": round(float(signal_current), 6),
        "bb_upper": round(float(bb_upper_current), 5),
        "bb_lower": round(float(bb_lower_current), 5),
        "bb_middle": round(float(ma_current), 5),
        "sma_fast": round(float(sma_fast), 5),
        "sma_slow": round(float(sma_slow), 5) if sma_slow is not None else
None
    }
    return indicators

# Main trading loop
try:
    while True:
        # Get recent bars (copy_rates_from_pos with index 0 for the current
```

```
ongoing bar)
        rates = mt5.copy_rates_from_pos(SYMBOL, TIMEFRAME, 0, BARS_COUNT)
        if rates is None:
            logging.error("Failed to retrieve data for %s, error: %s",
SYMBOL, mt5.last_error())
            time.sleep(60); continue  # wait and retry

        df = pd.DataFrame(rates)
        # Convert timestamp to datetime (not strictly needed for
calculations)
        df['time'] = pd.to_datetime(df['time'], unit='s')

        indicators = compute_indicators(df)
        if indicators is None:
            logging.warning("Not enough data to compute indicators, skipping
cycle")
            time.sleep(300); continue

        # Prepare the prompt for GPT-4
        latest_price = df['close'].iloc[-1]
        prompt = (f"Asset: {SYMBOL}, Timeframe: 5m\n"
                  f"Latest Price: {latest_price:.5f}\n"
                  f"Indicators:\n"
                  f"- RSI({RSI_PERIOD}): {indicators['rsi']}\n"
                  f"- MACD({FAST_MA},{SLOW_MA}) line: {indicators['macd']:.
6f}, signal: {indicators['macd_signal']:.6f}\n"
                  f"- Bollinger Bands({BB_PERIOD},{BB_STDDEV}):
upper={indicators['bb_upper']:.5f}, lower={indicators['bb_lower']:.5f}\n"
                  f"- SMA({SMA_FAST}): {indicators['sma_fast']:.5f},
SMA({SMA_SLOW}): {indicators['sma_slow']:.5f}\n\n"
                  "Question: Buy, Sell or Hold? Provide action, volume
(lots), stop-loss and take-profit.")

        # Call OpenAI API for trading decision
        try:
            api_response = openai.ChatCompletion.create(
                model="gpt-4",
                messages=[
                    {"role": "system", "content": "You are an expert trading
assistant. Analyze the indicators and price to give one trading action."},
                    {"role": "user", "content": prompt}
                ],
                temperature=0.0,
                max_tokens=100
            )
        except Exception as e:
            logging.error("OpenAI API call failed: %s", str(e))
            time.sleep(60); continue  # wait a minute and retry

        reply = api_response['choices'][0]['message']['content'].strip()
        logging.info("GPT-4 reply for %s: %s", SYMBOL, reply)
```

```
        # Parse the reply
        action = None; volume = None; sl_value = None; tp_value = None
        try:
            # Attempt to parse as JSON first (if GPT responded in JSON)
            import json
            decision = json.loads(reply)
            action = decision.get("action") or decision.get("Action")
            volume = decision.get("volume") or decision.get("Volume")
            sl_value = decision.get("stop_loss") or decision.get("StopLoss")
or decision.get("SL")
            tp_value = decision.get("take_profit") or
decision.get("TakeProfit") or decision.get("TP")
        except Exception:
            # Fallback to manual parsing if not JSON
            text = reply.lower()
            if "buy" in text: action = "buy"
            elif "sell" in text: action = "sell"
            elif "hold" in text or "no trade" in text: action = "hold"
            # Find numbers for volume, SL, TP
            # This simplistic approach finds the first occurrences of
patterns after keywords
            import re
            vol_match = re.search(r"volume[:\s]+([0-9]*\.?[0-9]+)", text)
            sl_match = re.search(r"stop[-\s]?loss[:\s]+([0-9]*\.?[0-9]+)",
text)
            tp_match = re.search(r"take[-\s]?profit[:\s]+([0-9]*\.?[0-9]+)",
text)
            if vol_match: volume = float(vol_match.group(1))
            if sl_match: sl_value = sl_match.group(1)
            if tp_match: tp_value = tp_match.group(1)

        if action is None:
            logging.warning("Could not parse action from GPT response: %s",
reply)
            time.sleep(300); continue

        action = action.lower()
        if action == "hold" or action == "no trade":
            logging.info("GPT-4 recommends hold/no trade for %s this cycle.",
SYMBOL)
            time.sleep(300); continue  # Skip trade execution

        # Default volume if not provided
        if volume is None:
            volume = 0.01  # minimal lot as fallback
        else:
            volume = float(volume)
            # Basic safety cap on volume
            if volume > 1.0:  # for example, cap to 1 lot for safety
                volume = 1.0
```

```
        # Determine current price and proper SL/TP values
        symbol_info_tick = mt5.symbol_info_tick(SYMBOL)
        if symbol_info_tick is None:
            logging.error("Failed to get tick info for %s", SYMBOL)
            time.sleep(300); continue
        current_bid = symbol_info_tick.bid
        current_ask = symbol_info_tick.ask
        # For a buy order, entry at ask; for sell, entry at bid
        entry_price = current_ask if action == "buy" else current_bid

        # Process stop-loss and take-profit
        sl_price = None; tp_price = None
        point = mt5.symbol_info(SYMBOL).point  # point size
        # If SL/TP values are given as string (could be "1.23456" or "50
pips")
        def parse_level(level, is_stop_loss=True):
            """Parse the level which might be price or pip distance."""
            level_str = str(level).strip().lower()
            if level_str.endswith("pip") or level_str.endswith("pips"):
                # It's given in pips
                try:
                    pips = float(re.findall(r"([0-9]*\.?[0-9]+)", level_str)
[0])
                except Exception:
                    return None
                # Convert pips to price difference
                # For forex, 1 pip = 0.0001 or 0.01 depending on symbol
digits.
                # Here we assume point is the smallest unit (e.g., 0.00001
for EURUSD)
                # and 1 pip = 10 points for 5-digit pricing.
                pip_value = pips * (0.0001 if mt5.symbol_info(SYMBOL).digits
in (5, 3) else 0.01)
                if action == "buy":
                    return entry_price - pip_value if is_stop_loss else
entry_price + pip_value
                else:  # sell
                    return entry_price + pip_value if is_stop_loss else
entry_price - pip_value
            else:
                # Assume it's an absolute price
                try:
                    price_val = float(level_str)
                except:
                    return None
                return price_val

        if sl_value:
            sl_price = parse_level(sl_value, is_stop_loss=True)
        if tp_value:
```

```
            tp_price = parse_level(tp_value, is_stop_loss=False)

        # If still None or not sensible, set default SL/TP distances (e.g.,
20 pips SL, 50 pips TP)
        if sl_price is None or tp_price is None:
            # Define a default pip distance
            default_sl_pips = 20; default_tp_pips = 50
            # Convert to price difference
            pip_unit = (0.0001 if mt5.symbol_info(SYMBOL).digits in (5,3)
else 0.01)
            if sl_price is None:
                sl_price = entry_price - default_sl_pips * pip_unit if action
== "buy" else entry_price + default_sl_pips * pip_unit
            if tp_price is None:
                tp_price = entry_price + default_tp_pips * pip_unit if action
== "buy" else entry_price - default_tp_pips * pip_unit
            logging.info("Using default SL/TP distances for %s: SL=%.5f,
TP=%.5f", SYMBOL, sl_price, tp_price)

        # Final safety check: SL < entry < TP for buys, or SL > entry > TP
for sells
        if action == "buy":
            if sl_price >= entry_price or tp_price <= entry_price:
                logging.warning("Discarding GPT signal due to invalid SL/TP
for buy: SL=%.5f, Entry=%.5f, TP=%.5f", sl_price, entry_price, tp_price)
                time.sleep(300); continue
        elif action == "sell":
            if sl_price <= entry_price or tp_price >= entry_price:
                logging.warning("Discarding GPT signal due to invalid SL/TP
for sell: SL=%.5f, Entry=%.5f, TP=%.5f", sl_price, entry_price, tp_price)
                time.sleep(300); continue

        # Prepare and send the trade request
        trade_type = mt5.ORDER_TYPE_BUY if action == "buy" else
mt5.ORDER_TYPE_SELL
        request = {
            "action": mt5.TRADE_ACTION_DEAL,
            "symbol": SYMBOL,
            "volume": volume,
            "type": trade_type,
            "price": entry_price,
            "sl": sl_price,
            "tp": tp_price,
            "deviation": 10,
            "magic": 123456,
            "comment": "GPT4_bot",
            "type_time": mt5.ORDER_TIME_GTC,
            "type_filling": mt5.ORDER_FILLING_RETURN,
        }
        result = mt5.order_send(request)
        if result.retcode != mt5.TRADE_RETCODE_DONE:
```

```
            logging.error("Trade execution failed: retcode=%s",
result.retcode)
        else:
            logging.info("Trade placed: %s %.2f lots at %.5f (SL=%.5f, TP=%.
5f)", action.upper(), volume, entry_price, sl_price, tp_price)

        # Wait for next cycle
        time.sleep(300)  # 5 minutes
finally:
    mt5.shutdown()
```

Script Explanation: This single-asset bot uses a continuous loop with a 5-minute sleep. Each iteration, it retrieves the latest M5 data for `SYMBOL`, computes indicators, and builds a prompt for GPT-4. The OpenAI response is parsed for action/volume/SL/TP and if a trade signal is given (buy/sell), a market order is sent with those parameters. It logs important events like API responses and trade outcomes. The code includes safety checks (e.g., ensuring SL and TP are on correct sides, capping volume, handling missing data). Always review the logs (`trading_bot.log`) to see the decisions and adjust the strategy or prompt as needed.

Portfolio Multi-Asset Trading Bot (`mt5_gpt_portfolio.py`)

This script extends the logic to multiple assets. It iterates through a list of symbols, querying each in turn. You could run this on, say, a list of currency pairs or a mix of asset classes. The structure is similar: for each symbol, get data, ask GPT-4, act on the response. We reuse the indicator calculation function for each symbol.

```
import os, time, logging
import MetaTrader5 as mt5
import pandas as pd

# Configuration
SYMBOLS = ["EURUSD", "GBPUSD", "USDJPY"]  # list of assets to monitor
TIMEFRAME = mt5.TIMEFRAME_M5
BARS_COUNT = 300
RSI_PERIOD = 14
FAST_MA = 12; SLOW_MA = 26; MACD_SIGNAL = 9
BB_PERIOD = 20; BB_STDDEV = 2
SMA_FAST = 50; SMA_SLOW = 200

logging.basicConfig(filename="trading_bot_portfolio.log",
                    level=logging.INFO,
                    format="%(asctime)s %(levelname)s %(message)s")
logging.info("Starting portfolio trading bot for symbols: %s", SYMBOLS)

if not mt5.initialize():
    logging.error("MT5 initialize() failed: %s", mt5.last_error())
    raise SystemExit("MT5 initialization failed")
# Ensure all symbols are available
for sym in SYMBOLS:
    info = mt5.symbol_info(sym)
```

```
    if info is None:
        logging.error("Symbol %s not found, exiting.", sym)
        mt5.shutdown()
        raise SystemExit(f"Symbol {sym} not found")
    if not info.visible:
        mt5.symbol_select(sym, True)

import openai
openai.api_key = os.getenv("OPENAI_API_KEY")
if openai.api_key is None:
    logging.error("OPENAI_API_KEY not set.")
    mt5.shutdown()
    raise SystemExit("Missing OpenAI API key")

def compute_indicators(df):
    # (Same as in single bot – omitted here for brevity, but essentially the
same code that returns the dict of indicators)
    # ...

    return indicators

try:
    while True:
        for sym in SYMBOLS:
            # Retrieve data for each symbol
            rates = mt5.copy_rates_from_pos(sym, TIMEFRAME, 0, BARS_COUNT)
            if rates is None:
                logging.error("No data for %s: %s", sym, mt5.last_error())
                continue  # skip this symbol this cycle
            df = pd.DataFrame(rates)
            df['time'] = pd.to_datetime(df['time'], unit='s')
            ind = compute_indicators(df)
            if ind is None:
                logging.warning("Not enough data to compute indicators for
%s", sym)
                continue

            latest_price = df['close'].iloc[-1]
            prompt = (f"Asset: {sym}, Timeframe: 5m\n"
                      f"Price: {latest_price:.5f}\n"
                      f"RSI({RSI_PERIOD}): {ind['rsi']}\n"
                      f"MACD({FAST_MA},{SLOW_MA}): {ind['macd']:.6f}, signal
{ind['macd_signal']:.6f}\n"
                      f"Bollinger({BB_PERIOD}): upper={ind['bb_upper']:.5f},
lower={ind['bb_lower']:.5f}\n"
                      f"SMA{SMA_FAST}: {ind['sma_fast']:.5f}, SMA{SMA_SLOW}:
{ind['sma_slow']:.5f}\n"
                      "Recommend: Buy, Sell or Hold? Provide volume, stop-
loss, take-profit.")
            try:
                api_response = openai.ChatCompletion.create(
```

```
                    model="gpt-4",
                    messages=[
                        {"role": "system", "content": "You are a trading bot
assistant. Analyze the given data and make a trade recommendation."},
                        {"role": "user", "content": prompt}
                    ],
                    temperature=0.0,
                    max_tokens=100
                )
            except Exception as e:
                logging.error("OpenAI API error for %s: %s", sym, e)
                continue
            reply = api_response['choices'][0]['message']['content'].strip()
            logging.info("GPT-4 reply for %s: %s", sym, reply)

            # Parse the reply (similar logic as single bot)...
            action = None; volume = None; sl_val = None; tp_val = None
            # Try JSON parse
            try:
                decision = json.loads(reply)
                action = decision.get("action") or decision.get("Action")
                volume = decision.get("volume")
                sl_val = decision.get("stop_loss") or decision.get("SL")
                tp_val = decision.get("take_profit") or decision.get("TP")
            except:
                text = reply.lower()
                if "buy" in text: action = "buy"
                elif "sell" in text: action = "sell"
                elif "hold" in text: action = "hold"
                vol_match = re.search(r"volume[:\s]+([0-9]*\.?[0-9]+)", text)
                sl_match = re.search(r"stop[-\s]?loss[:\s]+([0-9]*\.?
[0-9]+)", text)
                tp_match = re.search(r"take[-\s]?profit[:\s]+([0-9]*\.?
[0-9]+)", text)
                if vol_match: volume = float(vol_match.group(1))
                if sl_match: sl_val = sl_match.group(1)
                if tp_match: tp_val = tp_match.group(1)

            if action is None or action.lower() == "hold":
                logging.info("No trade for %s (action=%s)", sym, action)
                continue
            action = action.lower()
            volume = 0.01 if volume is None else float(volume)
            if volume <= 0: volume = 0.01
            if volume > 5: volume = 5.0  # cap max lots, for example

            # Get market prices
            tick = mt5.symbol_info_tick(sym)
            if not tick:
                logging.error("No tick info for %s", sym); continue
            bid, ask = tick.bid, tick.ask
```

```
            entry = ask if action == "buy" else bid
            point = mt5.symbol_info(sym).point

            # Convert SL/TP similar to single bot
            def parse_level(val, is_sl):
                s = str(val).lower()
                if "pip" in s:
                    try:
                        pips = float(re.findall(r"([0-9]*\.?[0-9]+)", s)[0])
                    except:
                        return None
                    pip_unit = 0.0001 if mt5.symbol_info(sym).digits in (3,5)
else 0.01
                    if action == "buy":
                        return entry - pips * pip_unit if is_sl else entry +
pips * pip_unit
                    else:
                        return entry + pips * pip_unit if is_sl else entry -
pips * pip_unit
                else:
                    try:
                        return float(s)
                    except:
                        return None
            sl_price = parse_level(sl_val, True) if sl_val else None
            tp_price = parse_level(tp_val, False) if tp_val else None
            # Apply default distances if needed (e.g., 20 pips SL, 50 pips
TP)
            if sl_price is None:
                sl_price = entry - (0.0001*20 if action=="buy" else
-0.0001*20)
            if tp_price is None:
                tp_price = entry + (0.0001*50 if action=="buy" else
-0.0001*50)

            # Validate SL/TP relative to entry
            if action == "buy":
                if sl_price >= entry or tp_price <= entry:
                    logging.warning("Skip %s trade, SL/TP invalid (SL=%.5f,
Entry=%.5f, TP=%.5f)", sym, sl_price, entry, tp_price)
                    continue
            if action == "sell":
                if sl_price <= entry or tp_price >= entry:
                    logging.warning("Skip %s trade, SL/TP invalid (SL=%.5f,
Entry=%.5f, TP=%.5f)", sym, sl_price, entry, tp_price)
                    continue

            # Place order
            order_type = mt5.ORDER_TYPE_BUY if action == "buy" else
mt5.ORDER_TYPE_SELL
            request = {
```

```
                "action": mt5.TRADE_ACTION_DEAL,
                "symbol": sym,
                "volume": volume,
                "type": order_type,
                "price": entry,
                "sl": sl_price,
                "tp": tp_price,
                "deviation": 20,
                "magic": 789012,
                "comment": "GPT4_bot",
                "type_time": mt5.ORDER_TIME_GTC,
                "type_filling": mt5.ORDER_FILLING_RETURN
            }
            result = mt5.order_send(request)
            if result.retcode != mt5.TRADE_RETCODE_DONE:
                logging.error("Order send failed for %s: retcode=%s", sym,
result.retcode)
            else:
                logging.info("%s -> Placed %s %.2f lots @%.5f (SL=%.5f TP=%.
5f)", sym, action.upper(), volume, entry, sl_price, tp_price)
        # End of symbol loop

        # Wait 5 minutes before next portfolio scan
        time.sleep(300)
finally:
    mt5.shutdown()
```

Script Explanation: The portfolio bot loops over each symbol in `SYMBOLS`. It fetches data and computes indicators for each, then queries GPT-4 individually per asset. (This ensures the prompt and reply remain focused on one symbol at a time, simplifying parsing and the model's task.) If a trade signal is given, it executes the trade similarly to the single bot. Each symbol's activity is logged in a separate log file (`trading_bot_portfolio.log`). The same safety measures apply, e.g., skipping trades on "hold" signals or invalid outputs. You can expand the SYMBOLS list as needed, but remember that more symbols mean more API calls (one per symbol per cycle), so monitor your rate limits and adjust sleep or model as appropriate.

Conclusion

By combining MetaTrader5's trading capabilities with GPT-4's analytical prowess, these bots demonstrate a cutting-edge approach to automated trading. They continuously analyze market conditions using classical technical indicators and then apply an AI model to interpret those conditions and make decisions [9] [2]. The provided scripts are well-commented for understanding and can be further customized – for example, you might integrate position management (closing trades based on AI signals or time), additional safety rules, or even news/sentiment data into the prompt for a more complex strategy. Always prioritize security (of your API keys and funds) and thorough testing. With careful prompt tuning and risk management, GPT-4 can serve as a powerful decision aid in algorithmic trading, while the MT5 API handles the execution reliably [5].

Sources:

- MetaTrader5 Python integration documentation and examples (for data access and order placement) [8] [5].
- OpenAI API usage guidelines and best practices for secure key management [6] [7].
- Community Q&A on computing technical indicators in Python (using Pandas/TA-Lib outside of MT5) [3] [4].
- "MetaTrader5 Trading Bot with GPT-4" – an example project demonstrating the concept of using GPT-4 to generate trading signals with entry/exit levels [1] [2].

---

[1] [2] [9] GitHub - Tzigger/MT5_trading_bot: This Python project integrates MetaTrader5 with GPT-4 to generate automated trading signals. It analyzes OHLC and tick data to provide real-time BUY/SELL recommendations, including entry, stop loss, and take profit levels.
https://github.com/Tzigger/MT5_trading_bot

[3] [4] [8] Python MetaTrader5 indicators - Stack Overflow
https://stackoverflow.com/questions/60404229/python-metatrader5-indicators

[5] order_send - Python Integration - MQL5 Reference - Reference on algorithmic/automated trading language for MetaTrader 5
https://www.mql5.com/en/docs/python_metatrader5/mt5ordersend_py

[6] [7] [10] [11] How to Create and Use an OpenAI ChatGPT API Key | Codecademy
https://www.codecademy.com/article/creating-an-openai-api-key