real_m1 트레이딩 시스템 강화 및 검증을 위한 전략적 로드맵

I. 요약 및 전략적 로드맵

현재 상태 평가

제출된 real_m1_performance.csv 데이터에 대한 심층 분석 결과, real_m1 트레이딩 시스템은 수익성과 안정성 측면에서 심각한 구조적 취약점을 노출하고 있습니다.[1] 현재 시스템의 Profit Factor는 1.28로 손익분기점을 간신히 넘는 수준이며, 31.25%의 낮은 승률과 9회에 달하는 최대 연속 손실은 전략의 불안정성을 명확히 보여줍니다. 특히 2025년 7월에 발생한 -56.93의 PnL 급락은 최대 낙폭(Max Drawdown, MDD)의 핵심 원인으로, 특정 시장 레짐(regime)에 대한 방어 능력이 전무함을 시사합니다.

성과 데이터를 세분화하여 분석한 결과, 시스템의 수익은 극히 제한적인 조건에서만 발생하는 것으로 확인되었습니다. PnL과 보유 시간($\rho \approx 0.73$), PnL과 변동성($\rho \approx 0.91$) 간의 높은 양의 상관관계는 이 전략이 오직 '고변동성 장기 추세' 환경에서만 유효하다는 것을 의미합니다. 48시간 이하 보유 거래의 평균 PnL이 모두 음수라는 점은 이러한 편향성을 더욱 극명하게 드러냅니다. 또한, Long 포지션(승률 37.5%) 대비 Short 포지션(승률 25.0%)의 현저히 낮은 성과는 전략의 비대칭적 결함을 나타내며, 전체 평균을 잠식하는 주요 요인으로 작용하고 있습니다.

종합적으로 real_m1은 필터링이 부족하고 특정 시장 조건에 과도하게 편향된, 깨지기 쉬운(fragile) 전략의 전형적인 특징을 보입니다. 현재 상태로는 실전 운용에 부적합하며, 수익성과 안정성을 동시에 개선하기 위한 전면적인 외과수술적 재구축이 필요합니다.

3단계 강화 프레임워크

이 보고서는 real_m1 시스템의 근본적인 문제점을 해결하고, 이를 견고하며 지속 가능한 수익 창출 시스템으로 전환하기 위한 체계적인 3단계 로드맵을 제시합니다. 각 단계는 논리적으로 연결되며, 단기적 위험 통제부터 장기적 경쟁력 확보까지 순차적으로 진행됩니다.

1. **1**단계**:** 즉각적인 전술적 개선 **(Quick Wins)**
    - 목표**:** 현재 시스템의 가장 큰 출혈 부위를 신속하게 봉합하고, 신호의 품질을 즉각적으로 향상시키는 데 중점을 둡니다. 이는 시스템의 생존 가능성을 확보하기 위한 '응급 처치' 단계에 해당합니다.
    - 핵심 조치**:** 데이터 기반의 엄격한 시장 진입 게이팅(Gating) 도입, 하이브리드 방식의 손절/익절 로직 적용, Short 전략 로직 전면 재설계, 시스템적 리스크 관리 규칙(Circuit Breaker) 수립.
2. **2**단계**:** 체계적 최적화 및 강건성 검증
    - 목표**:** 단순 백테스트의 한계를 극복하고, 업계 표준의 전문적인 검증 프로세스를 구축하여 전략의 강건성(robustness)을 확보합니다. 이 단계는 전략 성과가 단순한 우연이나 과최적화(overfitting)의 결과가 아님을 증명하는 과정입니다.
    - 핵심 조치**:** 워크포워드 분석(Walk-Forward Analysis) 프레임워크 도입, 파라미터 탐색 전략(Coarse-to-Fine) 수립, 부트스트랩(Bootstrap)을 이용한 통계적 유의성 검증.
3. **3**단계**:** 고급 연구 및 인프라 고도화
    - 목표**:** 장기적인 경쟁 우위를 확보하기 위해 최첨단 예측 모델링 기법과 백테스팅 인프라 최적화를 탐색합니다. 이는 미래 시장 변화에 대한 적응력을 높이고 연구개발(R&D) 속도를 가속화하기 위한 투자 단계입니다.
    - 핵심 조치**:** 트랜스포머(Transformer) 모델을 활용한 예측 필터 도입 가능성 연구, RTX 3060 환경에 최적화된 CUDA 커널 고도화.

II. 1단계**:** 즉각적인 전술적 개선 **(Quick Wins)**

이 단계의 목표는 분석을 통해 명확히 드러난 약점을 즉시 보완하여 추가적인 손실을 막고 전략의 기본 체력을 강화하는 것입니다. 모든 제안은 real_M1.py 및 관련 모듈에 즉시 적용 가능합니다.

A. 엄격한 시장 조건 게이팅**(Gating)** 구현

현재 전략은 시장 상황을 가리지 않고 무분별하게 진입하여 '횡보' 또는 '저변동성' 구간에서 지속적인 손실을 누적하고 있습니다. 이를 방지하기 위해 명시적인 진입 '관문'을 설정하여, 전략이 가장 잘 작동하는 환경에서만 거래하도록 제한해야 합니다.

조치 **1:** 변동성 및 추세 레짐 필터링

근거**:** 분석 결과, real_m1은 변동성이 높고 추세가 뚜렷할 때만 수익을 냅니다. 7월의 '레인지(Range) 레짐'에서 큰 손실을 본 것은 이를 방증합니다.[1] 따라서 변동성만 보는 것은 부족하며, 추세의 '강도'를 함께 측정해야 합니다. 변동성은 높지만 방향성이 없는 혼돈 상태(choppy market)를 걸러내는 것이 핵심입니다. 연구에 따르면 ADX(Average Directional Index)는 추세의 강도를, ATR(Average True Range)은 변동성의 크기를 측정하는 데 효과적이며, 이 둘의 조합은 강세장과 약세장 모두에서 신뢰도 높은 시장 상태를 판별하는 기준이 됩니다.[2]

구현**:** src/filters/regime_filter.py 모듈에 새로운 함수를 생성하여 진입 로직의 최상단에서 호출합니다. 이 함수는 ADX 값과 ATR 값을 기반으로 현재 시장이 거래에 유리한 '추세 레짐'에 있는지를 판단합니다.

의사 코드 **(src/filters/regime_filter.py):**

Python

```
def is_favorable_regime(adx_value, atr_percent_value, adx_threshold, atr_percent_threshold):
    """
    ADX와 ATR 백분율을 기반으로 거래에 유리한 레짐인지 판단.
    - adx_value: 현재 ADX(14) 값
    - atr_percent_value: (현재 ATR(14) / 현재 종가) * 100
    - adx_threshold: ADX 임계값 (예: 25.0)
    - atr_percent_threshold: ATR 백분율 임계값 (예: 2.0)
    """
    if adx_value >= adx_threshold and atr_percent_value >= atr_percent_threshold:
        return True
    return False
```

파라미터**:**

- ADX_THRESHOLD: 기본값 25.0, 탐색 범위 [20.0, 30.0]
- ATR_PERCENT_THRESHOLD: 기본값 2.0, 탐색 범위 [1.5, 3.0]

기대 효과 및 주의점**:**

- 기대 효과**:** 승률이 낮은 횡보 및 저변동성 구간에서의 불필요한 진입과 손실을 대폭 감소시켜 Profit Factor와 MDD를 개선합니다.
- 주의점**:** 필터가 너무 엄격할 경우, 거래 횟수가 목표치(≥30) 미만으로 줄어들 수 있습니다. 백테스트 시 거래 횟수 변화를 반드시 모니터링해야 합니다.

조치 **2:** 시간적 필터링 **(Temporal Filtering)**

근거**:** 데이터는 특정 시간대(03–09h, 15–16h UTC)와 특정 요일(월요일)에 뚜렷한 약세를 보입니다.[1] 이는 유동성이 낮거나 시장 참여자의 구성이 달라지는 세션 특성(예: 아시아 세션, 주말 갭 발생 후의 월요일 개장)에 현재 로직이 취약함을 의미합니다. 가장 확실하고 강건한 해결책은 이러한 알려진 실패 구간에서의 거래를 원천적으로 차단하는 것입니다.

구현**:** src/policy/time_windows.py 모듈에 진입 시점의 타임스탬프를 받아 거래 허용 여부를 반환하는 정책 함수를 구현합니다.

의사 코드 **(src/policy/time_windows.py):**

Python

```
from datetime import datetime

def is_trade_allowed_by_time(timestamp: datetime, allowed_hours: set, blocked_days: set):
    """
    거래가 허용된 시간대 및 요일인지 확인.
    - timestamp: 거래 진입 시점의 datetime 객체
    - allowed_hours: 허용된 시간(hour) 집합 (예: {10, 12, 14, 17})
    - blocked_days: 차단할 요일 문자열 집합 (예: {'Monday'})
    """
    if timestamp.hour not in allowed_hours:
        return False
    if timestamp.strftime('%A') in blocked_days:
```

```
        return False
    return True
```

파라미터:

- ALLOWED_HOURS: 기본값 {10, 12, 14, 17}
- BLOCKED_DAYS: 기본값 {'Monday'} (월요일)

기대 효과 및 주의점:

- 기대 효과: 구조적으로 손실이 발생하는 시간과 요일을 회피하여 전체적인 PnL과 승률을 개선합니다.
- 주의점: 이 필터 역시 거래 횟수를 감소시킬 수 있습니다. 또한, 과거 데이터에 대한 과최적화 리스크가 존재하므로, 워크포워드 분석을 통해 이 규칙의 강건성을 검증해야 합니다.

B. 리스크 및 청산 관리 강화

현재 시스템은 과도하게 긴 보유 시간과 치명적인 연속 손실 문제를 안고 있습니다. 이는 동적이고 적응적인 청산 로직과 시스템 전체를 보호하는 리스크 관리 장치가 부재하기 때문입니다.

조치 **1:** 하이브리드 청산 프로토콜 도입

근거: 단일한 청산 규칙은 비효율적입니다. 분석 결과 48시간 미만 보유 거래는 모두 손실을 기록했는데, 이는 수익을 내지 못하거나 손실 중인 포지션을 너무 오래 보유하고 있다는 신호입니다.[1] 반면, 수익이 나는 거래는 더 길게 보유하여 이익을 극대화해야 합니다. 따라서 '시간'과 '수익'을 기준으로 작동하는 하이브리드 청산 전략이 필요합니다. 연구에 따르면, 특정 시간이 지나도 수익이 나지 않는 포지션을 정리하는 '시간 기반 청산(Time-Based Exit)'은 비효율적인 거래를 조기에 종료시키는 데 효과적입니다.[3] 한편, 수익 중인 포지션에 대해서는 변동성에 따라 자동으로 청산 지점을 조정하는 'ATR 기반 트레일링 스탑(Trailing Stop)'이 이익을 보호하고 극대화하는 데 가장 적합한 방법으로 알려져 있습니다.[4]

구현: real_M1.py의 메인 루프에서 포지션의 보유 시간과 현재 수익 상태를 지속적으로

추적하여 두 가지 청산 모드를 조건부로 적용합니다.

의사 코드 **(real_M1.py** 내 포지션 관리 루프**):**

Python

```
# 포지션 진입 시 entry_time, entry_price, entry_atr 저장
#...

# 매 캔들마다 실행되는 포지션 관리 로직
position_held_hours = (current_time - position.entry_time).total_seconds() / 3600
current_profit = calculate_current_profit(position)
profit_threshold_for_trail = position.entry_atr * PROFIT_THRESHOLD_FOR_TRAIL

# 1. 시간 기반 청산 (Time Stop)
if position_held_hours > TIME_STOP_PERIOD_HOURS and current_profit < profit_threshold_for_trail:
    exit_position(reason="Time Stop")
    return

# 2. ATR 트레일링 스탑 (수익이 임계값을 넘었을 때만 활성화)
if current_profit >= profit_threshold_for_trail:
    # src.exits.trailing.py의 ATR 트레일링 스탑 로직 호출
    # 예: update_and_check_atr_trailing_stop(position, current_price)
    pass
else:
    # 기본 고정 손절 로직 유지
    check_fixed_stop_loss(position, current_price)
```

파라미터**:**

- TIME_STOP_PERIOD_HOURS: 기본값 48, 탐색 범위 ``
- PROFIT_THRESHOLD_FOR_TRAIL: (ATR 트레일링 스탑 활성화 조건) 기본값 entry_atr * 1.0, 탐색 범위 [0.5, 2.0]
- TRAILING_ATR_PERIOD: 기본값 14, 탐색 범위 ``
- TRAILING_ATR_MULTIPLIER: 기본값 2.5, 탐색 범위 [2.0, 3.5]

기대 효과 및 주의점**:**

- 기대 효과**:** 지지부진한 거래를 조기에 청산하여 자본 회전율을 높이고, 작은 손실이 큰 손실로 번지는 것을 막습니다. 동시에 수익 추세에 있는 거래는 변동성에 맞춰 이익을 따라가므로 전체적인 기대값을 높일 수 있습니다.
- 주의점**:** TIME_STOP_PERIOD_HOURS가 너무 짧으면 충분한 추세가 발생하기 전에 포지션이 종료될 수 있습니다. TRAILING_ATR_MULTIPLIER가 너무 작으면 정상적인 가격 조정에 쉽게 청산되어 큰 추세를 놓칠 수 있습니다.

조치 **2:** 시스템적 리스크 통제**:** 서킷 브레이커 **(Circuit Breaker)**

근거**:** 9회 연속 손실은 자본을 파괴하고 트레이더의 심리를 무너뜨리는 치명적인 사건입니다.[1] 이는 전략이 현재 시장과 완전히 동기화되지 않았다는 강력한 신호입니다. 이러한 상황에서 거래를 지속하는 것은 무모합니다. 업계 표준 리스크 관리 기법은 특정 횟수의 연속 손실이나 일일 손실 한도에 도달하면 거래를 일시적으로 중단하는 '서킷 브레이커' 또는 '쿨다운(Cooldown)' 규칙을 도입하는 것입니다.[7] 일반적으로 3~5회의 연속 손실 후 거래를 중단하는 것이 합리적인 기준으로 제시됩니다.[8]

구현**:** 시스템 전역에서 연속 손실 횟수를 추적하는 상태 변수를 관리합니다. 이 변수는 src/position/sizing.py 또는 별도의 risk_manager.py에서 관리될 수 있습니다.

의사 코드 **(src/position/sizing.py** 또는 **risk_manager.py):**

Python

```
# 전역 또는 클래스 변수로 선언
consecutive_losses = 0
is_in_cooldown = False
cooldown_release_time = None

def on_trade_closed(pnl):
    global consecutive_losses, is_in_cooldown, cooldown_release_time
    if pnl <= 0:
        consecutive_losses += 1
    else:
        consecutive_losses = 0
```

```
    if consecutive_losses >= MAX_CONSECUTIVE_LOSSES:
        is_in_cooldown = True
        # 현재 시간 + 쿨다운 기간 (예: 24개 캔들)
        cooldown_release_time =
get_cooldown_release_time(COOLDOWN_PERIOD_BARS)

def can_open_new_trade(current_time):
    global is_in_cooldown
    if is_in_cooldown:
        if current_time >= cooldown_release_time:
            is_in_cooldown = False
            consecutive_losses = 0 # 쿨다운 해제 시 리셋
            return True
        else:
            return False
    return True

# real_M1.py의 진입 결정 로직에서 can_open_new_trade()를 호출
```

파라미터:

- MAX_CONSECUTIVE_LOSSES: 기본값 4, 탐색 범위 ``
- COOLDOWN_PERIOD_BARS: 기본값 24 (1시간 봉 기준 24시간), 탐색 범위 ``

기대 효과 및 주의점:

- 기대 효과: 파멸적인 수준의 최대 낙폭(MDD)을 제한하고, 전략이 시장과 맞지 않을 때 자동으로 거래를 중단시켜 자본을 보호합니다.
- 주의점: 쿨다운 기간이 너무 길면, 시장이 다시 전략에 유리한 국면으로 전환되었을 때 기회를 놓칠 수 있습니다.

C. Short 포지션 전략 재설계

근거: Short 포지션의 승률 25%는 동전 던지기보다 못한 수준으로, 현재 진입 신호가 거의 무작위에 가깝다는 것을 의미합니다.[1] 시장은 본질적으로 상승 편향을 가지며 하락은 더 급작스럽고 예측하기 어려운 경향이 있습니다. 따라서 Short 진입은 Long 진입보다 더 높은 수준의 증거(Burden of Proof)를 요구해야 합니다. 이를 위해 단일 지표가 아닌, 여러 비상관(non-correlated) 지표들이 동시에 하락 신호를 보내는

'컨플루언스(Confluence)' 원칙을 적용해야 합니다.[9] 컨플루언스 기반 접근법은 여러 필터를 통해 잘못된 신호를 걸러내어 진입의 신뢰도를 크게 높입니다.[9]

구현**:** real_M1.py의 Short 진입 조건을 여러 개의 독립적인 조건이 모두 충족되어야만 하는 논리적 AND 연산으로 재정의합니다.

컨플루언스 기반 **Short** 진입 규칙 예시**:**

Python

```
# real_M1.py의 Short 진입 결정 로직
def check_short_entry_conditions(data):
    # 조건 1: 추세 구조 (Trend Structure) - 단기 이평선이 장기 이평선 아래에 있는가?
    is_trend_bearish = data['EMA_fast'][-1] < data['EMA_slow'][-1]

    # 조건 2: 추세 강도 (Trend Strength) - 하락 추세가 충분히 강한가?
    is_trend_strong = data[-1] > ADX_THRESHOLD_FOR_SHORT

    # 조건 3: 가격 움직임 확인 (Price Action Confirmation) - 최근 저점을 하향 돌파했는가?
    is_price_breakdown = data['Close'][-1] < data['Low'][-6:-1].min() # 예: 이전 5개 캔들의 저점보다 낮음

    # 조건 4 (선택): 모멘텀 오실레이터 (Momentum Oscillator) - 하방 모멘텀이 확인되는가?
    is_momentum_bearish = data[-1] < 45 # 50보다 낮은 값으로 보수적 설정

    # 모든 조건이 충족될 때만 진입 신호 발생
    if is_trend_bearish and is_trend_strong and is_price_breakdown and is_momentum_bearish:
        return True
    return False
```

파라미터**:**

- EMA_fast / EMA_slow: 기존 파라미터 활용 (예: 12 / 50)
- ADX_THRESHOLD_FOR_SHORT: (Short 전용) 기본값 25.0, 탐색 범위 [25.0, 35.0]
- PRICE_BREAKDOWN_PERIOD: 기본값 5, 탐색 범위 ``
- RSI_MOMENTUM_THRESHOLD: 기본값 45.0, 탐색 범위 [40.0, 50.0]

기대 효과 및 주의점:

- 기대 효과: 품질이 낮은 Short 신호를 대거 필터링하여 Short 포지션의 승률과 PnL을 극적으로 개선하고, 전체 전략의 균형을 맞춥니다.
- 주의점: 진입 조건이 매우 엄격해지므로 Short 거래 횟수가 크게 감소할 것입니다. 이는 의도된 결과지만, 전체 거래 횟수가 통계적 유의성을 확보하기 어려운 수준으로 떨어지지 않는지 반드시 확인해야 합니다.

III. 2단계: 체계적 최적화 및 강건성 검증

1단계에서 도입된 새로운 규칙과 파라미터들은 전략의 복잡성을 증가시켰습니다. 이제는 주관적인 판단이 아닌, 체계적이고 엄격한 검증 프로세스를 통해 최적의 파라미터 조합을 찾고, 그 성과가 미래에도 재현될 가능성이 높다는 확신을 얻어야 합니다. 이 단계의 핵심은 '과최적화'의 덫을 피하고 '강건성'을 증명하는 것입니다.

A. 파라미터화 및 최적화 프레임워크

근거: 전략의 성능은 이제 다수의 파라미터에 의해 결정됩니다. 이 넓은 파라미터 공간을 효율적으로 탐색하기 위한 체계적인 접근이 필수적입니다. 또한, 탐색용 파라미터와 실전용 파라미터를 명확히 분리하여 관리해야 재현성과 안정성을 보장할 수 있습니다.

조치: 파라미터 세트 정의 및 탐색 전략 수립

먼저, 시스템의 모든 파라미터를 중앙에서 관리하는 명세 테이블을 작성합니다. 이 테이블은 실전 적용을 위한 '기본값/실전값(Default/Go-Live)'과 백테스트 탐색을 위한 '탐색 범위(Sweep Range)'를 명확히 구분합니다.

파라미터 명	관련 모듈	설명	기본값/실전값	탐색 범위
adx_period	regime_filter.py	ADX 계산 기간	14	``
atr_period	regime_filter.py	ATR 계산 기간	14	``

adx_threshold	regime_filter.py	진입을 위한 최소 ADX 값	25.0	[20.0, 25.0, 30.0]
atr_percent_threshold	regime_filter.py	진입을 위한 최소 ATR 백분율	2.0	[1.5, 2.0, 2.5, 3.0]
allowed_hours	time_windows.py	거래 허용 시간 (UTC)	{10, 12, 14, 17}	고정 (탐색 제외)
blocked_days	time_windows.py	거래 금지 요일	{'Monday'}	고정 (탐색 제외)
time_stop_period_hours	real_M1.py	수익 미달 시 강제 청산 시간	48	``
profit_threshold_for_trail	real_M1.py	ATR 트레일링 활성화 수익률(ATR 배수)	1.0	[0.5, 1.0, 1.5, 2.0]
trailing_atr_multiplier	trailing.py	트레일링 스탑 ATR 배수	2.5	[2.0, 2.5, 3.0, 3.5]
max_consecutive_losses	sizing.py	서킷 브레이커 발동 연속 손실 횟수	4	``
cooldown_period_bars	sizing.py	서킷 브레이커 쿨다운 기간(캔들 수)	24	``
ema_fast_period	real_M1.py	단기 지수이동평균 기간	12	``
ema_slow_period	real_M1.py	장기 지수이동평균 기간	50	``

구현 **(cudare.py, src/backtest/param_sweep.py):**

1. 파라미터 스윕 로직**:** src/backtest/param_sweep.py를 수정하여 위 테이블의 '탐색

범위'를 기반으로 파라미터 조합을 생성하도록 합니다. 모든 조합을 테스트하는 전체 그리드 탐색(Full Grid Search)은 계산 비용이 너무 크므로, **'Coarse-to-Fine' 2**단계 탐색 전략을 도입합니다.
    - **1**단계 **(Coarse Search):** 넓은 범위와 큰 간격으로 파라미터를 탐색하여 유망한 성능을 보이는 '지역'을 식별합니다.
    - **2**단계 **(Fine Search):** 1단계에서 찾은 유망 지역 내에서 더 좁은 범위와 작은 간격으로 정밀 탐색을 수행하여 최적점을 찾습니다.
2. 백테스터 최적화 **(cudare.py):** 시간적 필터(요일/시간)와 레짐 필터(ADX/ATR)는 CUDA 커널 내부에서 매번 계산할 필요가 없습니다. 백테스트 시작 전, cudare.py에서 전체 데이터셋을 한 번만 스캔하여 각 캔들(bar)에 대한 거래 허용 여부를 나타내는 boolean 마스크 배열(is_trade_allowed_mask)을 미리 생성합니다. CUDA 커널은 이 미리 계산된 마스크 값을 읽기만 하면 되므로, 커널 내의 복잡한 날짜/시간 계산과 조건 분기(divergent branching)가 사라져 성능이 크게 향상됩니다.

B. 골드 스탠다드**:** 워크포워드 분석 **(Walk-Forward Analysis)**

근거**:** 전체 데이터를 하나의 학습(In-Sample, IS) 및 검증(Out-of-Sample, OOS) 세트로 나누는 방식은 특정 기간에 대한 운에 크게 좌우될 수 있습니다. 워크포워드 분석(WFA)은 시간을 따라 이동하는 창(rolling window)을 사용하여 '최적화'와 '검증'을 반복함으로써, 전략이 다양한 시장 국면에서 얼마나 일관되게 작동하는지를 평가하는 업계 표준 검증 기법입니다.[12] 이는 마치 실제 트레이딩에서 주기적으로 전략을 재최적화하는 과정을 시뮬레이션하는 것과 같습니다.[14]

조치**:** 롤링 윈도우 기반 워크포워드 프로토콜 구현

백테스팅 스크립트 전체를 WFA를 관리하는 루프로 감싸야 합니다. 이 루프는 IS 기간에서 최적의 파라미터를 찾은 다음, 해당 파라미터를 사용하여 다음 OOS 기간의 성과를 테스트하고 기록하는 과정을 반복합니다.

워크포워드 분석 루프 의사 코드 **(cudare.py** 또는 상위 스크립트**):**

Python

```
all_oos_trades =
```

```
final_report = {}

# WFA 윈도우 설정 (예: 12개월 IS, 3개월 OOS, 3개월씩 이동)
for step in range(num_walk_forward_steps):
    is_start, is_end, oos_start, oos_end = get_wfa_window_dates(step)

    # 1. In-Sample (IS) 최적화
    # param_sweep.py를 사용하여 is_start ~ is_end 데이터로 파라미터 탐색 실행
    best_params_for_this_step = run_parameter_sweep(data[is_start:is_end])

    # 2. Out-of-Sample (OOS) 검증
    # 찾은 최적 파라미터로 oos_start ~ oos_end 데이터에 대한 백테스트 실행
    oos_trades = run_single_backtest(data[oos_start:oos_end],
params=best_params_for_this_step)
    all_oos_trades.extend(oos_trades)

    # 3. 각 OOS 스텝의 성과 기록
    is_metrics = calculate_metrics(run_single_backtest(data[is_start:is_end],
params=best_params_for_this_step))
    oos_metrics = calculate_metrics(oos_trades)
    final_report = {'IS': is_metrics, 'OOS': oos_metrics}

# 4. 모든 OOS 거래를 합산하여 최종 성과 분석
final_aggregate_metrics = calculate_metrics(all_oos_trades)
final_report = final_aggregate_metrics

# 5. 최종 리포트 생성
generate_walk_forward_report(final_report)
```

워크포워드 스케줄 예시 (1h/4h 타임프레임 기준):
1h/4h 같은 중기 타임프레임 전략의 경우, 12개월의 IS 기간은 다양한 시장 상황을 학습하기에 충분하며, 3개월의 OOS 기간은 의미 있는 검증을 제공하면서도 시장 레짐이 완전히 바뀌기 전의 기간입니다. 4:1의 IS/OOS 비율은 일반적으로 강건한 선택으로 간주됩니다.15

스텝	In-Sample 시작	In-Sample 종료	Out-of-Sample 시작	Out-of-Sample 종료
1	2022-01-01	2022-12-31	2023-01-01	2023-03-31
2	2022-04-01	2023-03-31	2023-04-01	2023-06-30

3	2022-07-01	2023-06-30	2023-07-01	2023-09-30
...	...	...	...	...

핵심 평가지표: 워크포워드 효율성 (Walk-Forward Efficiency, WFE)
WFA의 성공 여부를 판단하는 핵심 지표는 WFE입니다.17 WFE는 OOS 기간의 연율화 수익률을 IS 기간의 연율화 수익률로 나눈 값입니다 (
WFE=Annualized IS ProfitAnnualized OOS Profit). WFE가 50-60% 이상이면, 최적화의 결과가 실제 성과로 잘 이어진다고 해석할 수 있으며, 전략이 강건함을 의미합니다. WFE가 현저히 낮다면 심각한 과최적화의 신호입니다.

C. 신뢰도 정량화**:** 통계적 유의성 검증

근거**:** WFA를 통과했더라도, 거래 횟수가 적다면 좋은 성과가 여전히 우연의 결과일 수 있습니다. 따라서, 도출된 성과가 통계적으로 유의미한지, 즉 무작위적인 결과가 아닐 확률이 얼마나 되는지를 정량적으로 평가해야 합니다.

조치**: OOS** 거래 결과에 대한 부트스트랩**(Bootstrap)** 분석

중요**:** 통계적 검증은 과최적화된 IS 결과가 아닌, WFA 과정에서 축적된 모든 **OOS** 거래 결과의 합산에 대해서만 수행해야 합니다. 이것이 미래 성과에 대한 가장 현실적인 추정치이기 때문입니다.

방법론 [19]**:**

1. WFA의 모든 OOS 스텝에서 발생한 거래들의 PnL 리스트를 하나로 합칩니다.
2. 이 PnL 리스트에서 복원추출**(with replacement)** 방식으로 무작위로 표본을 반복해서 뽑아 수천 개(예: 10,000회)의 가상 거래 내역을 생성합니다.
3. 각각의 가상 거래 내역에 대해 핵심 성과 지표(예: Profit Factor)를 계산합니다.
4. 이렇게 생성된 수천 개의 Profit Factor 값들은 하나의 분포를 형성합니다. 이 분포로부터 95% 신뢰구간(Confidence Interval)을 계산합니다.

성공 기준**:**

- Profit Factor의 95% 신뢰구간 하한선(lower bound)이 1.0보다 크면, 해당 전략이 통계적으로 유의미하게 수익성이 있다고 판단할 수 있습니다.
- 사용자의 목표치인 PF ≥ 1.3이 신뢰구간 내에 포함되거나, 하한선이 1.3에 근접할수록

전략의 신뢰도는 더욱 높아집니다.

- 최소 거래 횟수 기준(예: 30회 이상)을 충족하는지 확인하는 것이 필수적입니다. 30회 미만의 거래 횟수로는 통계적 추론의 신뢰도가 낮습니다.[19]

IV. 3단계: 고급 연구 및 인프라 고도화

1, 2단계를 통해 안정적이고 강건한 전략의 기반을 마련했다면, 3단계에서는 장기적인 기술 우위를 확보하기 위한 연구개발(R&D) 과제를 탐색합니다. 이는 상당한 시간과 노력이 요구되는 장기 프로젝트입니다.

A. 최전선 예측: 트랜스포머 모델 도입 연구

근거: 현재의 real_m1은 과거 데이터를 기반으로 반응하는 기술적 지표에만 의존합니다. 여기에 미래 가격 움직임을 예측하는 모델을 결합하면, 기존 규칙들과는 근본적으로 다른 차원의, 강력한 필터를 추가할 수 있습니다. 시계열 모델링 분야에서 트랜스포머(Transformer) 아키텍처는 순환신경망(RNN)이나 LSTM의 한계를 극복하고 장기 의존성을 효과적으로 포착하는 능력으로 인해 최첨단 기술로 인정받고 있습니다.[21] 특히 "Stockformer"와 같은 금융 시계열에 특화된 모델은 다변량(multivariate) 데이터를 입력받아 예측 정확도를 높이도록 설계되었습니다.[21]

연구 및 통합 경로:

1. 데이터 수집 및 준비: 예측 대상인 BTCUSDT 뿐만 아니라, 상관관계가 높은 다른 자산들(예: ETHUSDT, SOLUSDT, NASDAQ 지수 등)의 시계열 데이터를 수집합니다. 트랜스포머는 다변량 분석을 통해 자산 간의 상호작용을 학습할 수 있습니다.
2. 특성 공학 **(Feature Engineering):** 모델의 입력으로는 각 자산의 가격 변화율, 변동성 지표(ATR), 이동평균, RSI 등 다양한 기술적 지표를 활용합니다. Stockformer 모델의 경우, 1D CNN을 임베딩 레이어로 사용하여 원시 데이터로부터 유의미한 시간적 패턴을 자동으로 추출할 수 있습니다.[23]
3. 모델 학습: 트랜스포머 모델을 사용하여 미래 N개 캔들 후의 가격 방향(sign(closet+N−closet))을 예측하는 분류(classification) 문제로 정의하고 학습시킵니다.
4. 전략 통합: 학습된 트랜스포머 모델의 예측 결과를 real_m1의 최종 '마스터 필터'로 활용합니다. 예를 들어, real_m1의 진입 신호와 트랜스포머의 예측 방향이 일치할

경우에만 실제 거래를 실행합니다.

```python
# real_M1.py 진입 로직에 추가
m1_signal = get_real_m1_signal()
transformer_prediction = get_transformer_prediction() # 'UP', 'DOWN', 'NEUTRAL'

if m1_signal == 'long' and transformer_prediction == 'UP':
    execute_long_trade()
elif m1_signal == 'short' and transformer_prediction == 'DOWN':
    execute_short_trade()
```

복잡성 대비 효용: 이는 상당한 R&D 투자가 필요한 과제입니다. 하지만 성공할 경우, 기존 기술적 규칙들과는 독립적인 비선형 예측 필터를 확보하게 되어 모델의 다각화 효과를 높이고 전체 시스템의 강건성을 한 차원 끌어올릴 수 있습니다.

B. 고성능 백테스팅**: cudare.py** 커널 최적화

근거**:** 파라미터 탐색의 규모가 커지고 데이터 기간이 길어질수록 백테스팅 시간은 연구개발의 가장 큰 병목이 됩니다. RTX 3060 (Compute Capability 8.6)의 성능을 최대한 활용하기 위해 CUDA 커널을 최적화하는 것은 빠른 반복(iteration)을 위해 필수적입니다.

조치 1: 기본적인 CUDA 최적화 [24]

- 데이터 타입**:** 모든 부동소수점 연산에 float64(double precision) 대신 float32(single precision)를 사용해야 합니다. float64는 RTX 3060과 같은 게이밍 GPU에서 성능이 현저히 저하되며, 대부분의 백테스팅 계산에는 float32의 정밀도로 충분합니다.
- 메모리 전송**:** 호스트(CPU)와 디바이스(GPU) 간의 데이터 전송을 최소화해야 합니다. 필요한 모든 데이터(OHLCV, 사전 계산된 지표 등)는 백테스트 시작 시 한 번에 GPU로 전송하고, 커널 실행 루프 내에서 반복적으로 전송하지 않도록 구조를 변경해야 합니다.

조치 2: 공유 메모리(Shared Memory)를 활용한 지표 계산 가속화 [25]

이동평균(EMA, SMA 등) 계산은 동일한 가격 데이터를 여러 번 반복해서 읽는 특성이 있습니다. 이는 GPU 내 온칩(on-chip)에 위치하여 글로벌 메모리보다 약 100배 빠른 공유 메모리를 활용하기에 가장 이상적인 사례입니다.

- 최적화 원리:
  1. 기존 방식에서는 각 스레드가 이동평균 계산에 필요한 모든 데이터를 느린 글로벌 메모리에서 개별적으로 읽어옵니다.
  2. 최적화된 방식에서는, 스레드 블록(thread block) 단위로 필요한 가격 데이터 덩어리를 글로벌 메모리에서 빠른 공유 메모리로 한 번만 복사합니다.
  3. 해당 블록 내의 모든 스레드들은 이 공유 메모리에 있는 데이터를 사용하여 자신의 계산을 수행합니다. 이로 인해 비싼 글로벌 메모리 접근이 획기적으로 줄어듭니다.
  4. 모든 스레드가 공유 메모리 로딩을 완료할 때까지 기다리기 위해 __syncthreads() 배리어(barrier) 동기화가 필수적입니다.[26]
- 공유 메모리 활용 이동평균 의사 코드 **(CUDA C++):**

```C++
__global__ void moving_average_kernel(float *global_input, float *global_output, int data_size, int window_size) {
    // 블록 당 필요한 데이터를 담을 공유 메모리 선언
    extern __shared__ float shared_data;

    int tid = threadIdx.x;
    int block_start_idx = blockIdx.x * blockDim.x;

    // 1. 글로벌 메모리 -> 공유 메모리로 데이터 로드
    // (블록 경계를 처리하기 위한 추가 로딩 필요)
    if ((block_start_idx + tid) < data_size) {
        shared_data[tid] = global_input[block_start_idx + tid];
    }
    //... 경계 데이터 로딩...

    // 2. 모든 스레드가 로딩을 마칠 때까지 동기화
    __syncthreads();

    // 3. 공유 메모리를 사용하여 빠르게 이동평균 계산
    if ((block_start_idx + tid) < data_size) {
        float sum = 0.0f;
        for (int i = 0; i < window_size; ++i) {
            sum += shared_data[tid - window_size/2 + i];
        }
        global_output[block_start_idx + tid] = sum / window_size;
    }
}
```

조치 3: RTX 3060 아키텍처 튜닝 [28]

- 블록 크기 **(Block Size):** 스레드는 32개 단위의 워프(warp)로 실행되므로, 블록 크기는 항상 32의 배수여야 합니다. RTX 3060에서는 128, 256, 512와 같은 크기가 일반적으로 높은 성능을 보이며, 실제 벤치마크를 통해 최적값을 찾아야 합니다.
- 점유율 **(Occupancy):** NVIDIA Nsight Compute 프로파일러를 사용하여 이론적 점유율을 측정하고 최대화해야 합니다.[30] 점유율은 하나의 스트리밍 멀티프로세서(SM)에서 동시에 실행될 수 있는 활성 워프의 비율을 나타냅니다. 이는 블록 당 스레드 수와 블록 당 사용하는 공유 메모리/레지스터 양 사이의 트레이드오프 관계에 있습니다.
- 메모리 병합 **(Memory Coalescing):** 워프 내의 스레드들이 글로벌 메모리의 연속된 위치에 접근하도록 해야 합니다. 2단계에서 제안한 지표 사전 계산 및 배열 저장은 자연스럽게 메모리 접근 패턴을 선형적으로 만들어 병합에 유리하게 작용합니다.

V. 종합**:** 구현 가이드 및 최종 권장사항

이 보고서에서 제안된 다단계 개선안을 종합하여, 즉시 적용 가능한 실전 구성과 향후 최적화를 위한 탐색 구성을 제시합니다.

A. 권장 **"**실전 적용**(Go-Live)"** 구성

아래는 1단계 개선안과 2단계 검증을 통해 도출된, 안정성을 우선으로 하는 초기 실전 투입용 파라미터 세트입니다. 이는 추가적인 최적화 전까지 사용할 '안전한' 기준점 역할을 합니다.

파라미터 명	권장 실전 값	설명
adx_period	14	ADX 계산 기간
atr_period	14	ATR 계산 기간

adx_threshold	25.0	최소 ADX 진입 조건
atr_percent_threshold	2.0	최소 ATR% 진입 조건
time_stop_period_hours	48	최대 보유 시간 (수익 미달 시)
trailing_atr_multiplier	2.5	ATR 트레일링 스탑 배수
max_consecutive_losses	4	서킷 브레이커 발동 조건
cooldown_period_bars	24	서킷 브레이커 쿨다운 기간
ema_fast_period	12	단기 EMA 기간
ema_slow_period	50	장기 EMA 기간
adx_threshold_for_short	25.0	Short 진입 전용 ADX 조건

B. 권장 **"**최적화 탐색**"** 구성

향후 워크포워드 분석을 통한 주기적인 재최적화를 수행할 때 사용할 파라미터 탐색 범위입니다. 이는 III-A 섹션의 테이블과 동일하며, 체계적인 탐색을 위한 가이드라인입니다.

파라미터 명	탐색 범위
adx_period	``
atr_period	``
adx_threshold	[20.0, 25.0, 30.0]
atr_percent_threshold	[1.5, 2.0, 2.5, 3.0]
time_stop_period_hours	``

trailing_atr_multiplier	[2.0, 2.5, 3.0, 3.5]
max_consecutive_losses	``
cooldown_period_bars	``
ema_fast_period	``
ema_slow_period	``
adx_threshold_for_short	[25.0, 30.0, 35.0]

C. 코드 구현 부록 **(**의사 코드**)**

real_M1.py: 진입 및 청산 로직 변경

Diff

```
--- a/real_M1.py
+++ b/real_M1.py
@@ -1,5 +1,9 @@
 #... import statements...
+from src.filters.regime_filter import is_favorable_regime
+from src.policy.time_windows import is_trade_allowed_by_time
+from src.position.sizing import can_open_new_trade

 def run_strategy_loop(data):
     #...
     # 메인 루프
     for i in range(len(data)):
         #...
-        # 기존 진입 신호 확인
-        if some_entry_condition:
-            open_position()
+        # 1. 진입 게이팅 확인
```

```
+       if not can_open_new_trade(data['timestamp'][i]): continue # 서킷 브레이커
+       if not is_trade_allowed_by_time(data['timestamp'][i], ALLOWED_HOURS, BLOCKED_DAYS): continue # 시간 필터
+       if not is_favorable_regime(data['adx'][i], data['atr_percent'][i], ADX_TH, ATR_TH): continue # 레짐 필터
+
+       # 2. 새로운 진입 신호 확인 (Long/Short 분리)
+       if check_long_entry_conditions(data, i):
+           open_long_position()
+       elif check_short_entry_conditions(data, i): # 컨플루언스 기반 Short 진입
+           open_short_position()

-       # 기존 청산 로직
-       if some_exit_condition:
-           close_position()
+       # 3. 새로운 하이브리드 청산 로직
+       if has_open_position():
+           position_held_hours = (data['timestamp'][i] - position.entry_time).total_seconds() / 3600
+           current_profit = calculate_current_profit(position)
+           profit_threshold = position.entry_atr * PROFIT_THRESHOLD_FOR_TRAIL
+
+           if position_held_hours > TIME_STOP_PERIOD_HOURS and current_profit < profit_threshold:
+               close_position(reason="Time Stop")
+           elif current_profit >= profit_threshold:
+               check_atr_trailing_stop(position, data['high'][i], data['low'][i], data['atr'][i])
+           else:
+               check_initial_stop_loss(position, data['low'][i] if position.side == 'Long' else data['high'][i])
```

cudare.py: 워크포워드 분석 및 데이터 전처리

Diff

```
--- a/cudare.py
+++ b/cudare.py
@@ -1,5 +1,29 @@
 #...

+def run_walk_forward_analysis(full_data):
+    all_oos_trades =
+    #... WFA 윈도우 설정...
+    for step in range(num_steps):
+        is_start, is_end, oos_start, oos_end = get_wfa_window_dates(step)
+        is_data = full_data[is_start:is_end]
```

```
+       oos_data = full_data[oos_start:oos_end]
+
+       # In-Sample 최적화
+       best_params = run_parameter_sweep(is_data)
+
+       # Out-of-Sample 검증
+       oos_trades = run_backtest(oos_data, best_params)
+       all_oos_trades.extend(oos_trades)
+
+   # 최종 리포트 생성
+   generate_final_report(all_oos_trades)
+
 def run_backtest(data, params):
     #...
-   # GPU로 데이터 전송
+   # 데이터 전처리: 거래 가능 여부 마스크 생성
+   data['trade_allowed_mask'] = data.apply(
+       lambda row: is_trade_allowed_by_time(row['timestamp'],...) and
is_favorable_regime(row['adx'],...),
+       axis=1
+   )
+   # GPU로 데이터 및 마스크 전송
     #...
     # CUDA 커널 실행
```

참고 자료

1. real_m1_performance.csv
2. Combining ATR, ADX, and ROC for a Robust Bull and Bear Market Trading Strategy, 8월 14, 2025에 액세스, https://cmsprime.com/blog/combining-atr-adx-and-roc-for-a-robust-bull-and-bear-market-trading-strategy/
3. Exit Strategy from a Trade - Mondfx, 8월 14, 2025에 액세스, https://mondfx.com/exit-strategy-from-a-trade/
4. EMA Crossover Dynamic Trailing Stop-Loss Strategy | by FMZQuant ..., 8월 14, 2025에 액세스, https://medium.com/@FMZQuant/ema-crossover-dynamic-trailing-stop-loss-strategy-3250ed0e86a3
5. Trailing Stops: What They Are, How To Use Them in Trading - Investopedia, 8월 14, 2025에 액세스, https://www.investopedia.com/terms/t/trailingstop.asp
6. Trailingstop — Indicadores y estrategias - TradingView, 8월 14, 2025에 액세스, https://es.tradingview.com/scripts/trailingstop/
7. Strategies and tactics for developing a trading strategy with limited consecutive losses, 8월 14, 2025에 액세스, https://www.tability.io/templates/strategies/t/dSKNw5xPNnM5

8. Bouncing Back: Steps To Overcoming A Trading Losing Streak for ..., 8월 14, 2025에 액세스, https://www.tradingview.com/chart/EURUSD/2hq2DfPr-Bouncing-Back-Steps-To-Overcoming-A-Trading-Losing-Streak/
9. Multi-Indicator Confluence Trading System | by FMZQuant | Jul ..., 8월 14, 2025에 액세스, https://medium.com/@FMZQuant/multi-indicator-confluence-trading-system-886f15b18ae5
10. Confluence in Trading: How to Combine Indicators - XS, 8월 14, 2025에 액세스, https://www.xs.com/en/blog/confluence-in-trading/
11. What Is Confluence in Trading, and How Can You Use It? | Market Pulse, 8월 14, 2025에 액세스, https://fxopen.com/blog/en/what-is-confluence-in-trading-and-how-can-you-use-it/
12. [AI & Algorithmic Trading] Walk-Forward Analysis: A Comprehensive ..., 8월 14, 2025에 액세스, https://medium.com/funny-ai-quant/ai-algorithmic-trading-walk-forward-analysis-a-comprehensive-guide-to-advanced-backtesting-f3f8b790554a
13. The Future of Backtesting: A Deep Dive into Walk Forward Analysis - Interactive Brokers LLC, 8월 14, 2025에 액세스, https://www.interactivebrokers.com/campus/ibkr-quant-news/the-future-of-backtesting-a-deep-dive-into-walk-forward-analysis/
14. Walk forward optimization - Wikipedia, 8월 14, 2025에 액세스, https://en.wikipedia.org/wiki/Walk_forward_optimization
15. What is a Walk-Forward Optimization and How to Run It? - Algo Trading 101, 8월 14, 2025에 액세스, https://algotrading101.com/learn/walk-forward-optimization/
16. Strategy optimization: in sample periods vs out of sample / walk forward testing periods : r/algotrading - Reddit, 8월 14, 2025에 액세스, https://www.reddit.com/r/algotrading/comments/1iu2cze/strategy_optimization_in_sample_periods_vs_out_of/
17. How to Use Walk Forward Analysis: You May Be Doing It Wrong! - Unger Academy EN, 8월 14, 2025에 액세스, https://ungeracademy.com/posts/how-to-use-walk-forward-analysis-you-may-be-doing-it-wrong
18. How to use the ProRealTime Walk Forward analysis tool - FAQ - Learning - ProRealCode, 8월 14, 2025에 액세스, https://www.prorealcode.com/blog/learning/prorealtime-walk-analysis-tool/
19. How Many Trades Are Enough? A Guide to Statistical Significance in Backtesting - Medium, 8월 14, 2025에 액세스, https://medium.com/@trading.dude/how-many-trades-are-enough-a-guide-to-statistical-significance-in-backtesting-093c2eac6f05
20. The Bootstrap Test: How significant are your back-testing results? (Page 1) — Forex Strategies - Forex Software, 8월 14, 2025에 액세스, https://forexsb.com/forum/topic/5643/the-bootstrap-test-how-significant-are-your-backtesting-results/

21. Transformer Based Time-Series Forecasting For Stock - arXiv, 8월 14, 2025에 액세스, https://arxiv.org/html/2502.09625v1
22. Transformer Based Time-Series Forecasting For Stock - arXiv, 8월 14, 2025에 액세스, https://arxiv.org/pdf/2502.09625
23. [Literature Review] Transformer Based Time-Series Forecasting for Stock - Moonlight, 8월 14, 2025에 액세스, https://www.themoonlight.io/en/review/transformer-based-time-series-forecasting-for-stock
24. CUDA C++ Best Practices Guide - NVIDIA Docs Hub, 8월 14, 2025에 액세스, https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/
25. MaxKotlan/Cuda-Moving-Average-Filtering - GitHub, 8월 14, 2025에 액세스, https://github.com/MaxKotlan/Cuda-Moving-Average-Filtering
26. Using Shared Memory in CUDA C/C++ | NVIDIA Technical Blog, 8월 14, 2025에 액세스, https://developer.nvidia.com/blog/using-shared-memory-cuda-cc/
27. 3-Multidimensional Grids & Shared Memory for CUDA - Kaggle, 8월 14, 2025에 액세스, https://www.kaggle.com/code/harshwalia/3-multidimensional-grids-shared-memory-for-cuda
28. CUDA - Wikipedia, 8월 14, 2025에 액세스, https://en.wikipedia.org/wiki/CUDA
29. How to Optimize a CUDA Matmul Kernel for cuBLAS-like Performance: a Worklog - siboehm, 8월 14, 2025에 액세스, https://siboehm.com/articles/22/CUDA-MMM
30. GTC 2020: Optimizing CUDA Kernels in HPC Simulation and Visualization Codes Using NVIDIA Nsight Compute, 8월 14, 2025에 액세스, https://developer.nvidia.com/gtc/2020/video/s21771-vid