ChatGPT

Diagnosing Post-Restart Order Submission Failures in ai-trading-bot

Executive summary

The most likely reason your bot "successfully submits orders immediately after a systemd restart but later (in the main loop) fails to submit orders" is that the repository's **live execution layer contains many "soft-fail" early-return guards** that *skip* order submission without crashing the process. These guards can be triggered only after the service has been running for a while because they depend on **state accumulated across loop iterations** (per-cycle caps, duplicate-intent windows, cached account snapshots, provider-disable flags, and broker lockouts). The first successful "post-restart" submit happens while these guards are still in their "fresh/empty" state (and — per the codebase — after a startup/warmup cycle is run). [1] [2]

Two repository-specific mechanisms are especially consistent with your symptom pattern:

- **Broker lockout after unauthorized responses**: the live execution engine can enter a timed suppression window after a 401/403-like broker error, and then **silently suppress subsequent submissions** for a cooldown period (default is 120 seconds in code). That yields "works right after restart, then later nothing gets submitted," especially if subsequent iterations encounter an auth-related failure. [2]
- **"Safety / quote quality / policy" gates** that return `None` (skip) rather than raising: safe-mode/disabled-provider, quote freshness / NBBO requirements, PDT restrictions, capacity prechecks, and per-cycle pacing caps can all stop orders from being sent even though upstream logic "decided" to trade. [2]

Separately, if your loop is making frequent broker/data calls, **rate limiting** can become a time-dependent failure mode; Alpaca Markets [3] documents throttling (e.g., 200 requests/minute/account) and 429 behavior. [4]

Because these are mostly "skip" behaviors rather than crashes, the key to diagnosis is to (1) **surface the specific skip reason** in logs and (2) confirm whether the "failure" is an exception (401/429/etc.) or "policy gating" (safe mode, quote quality, PDT, pacing, duplicates).

Repository map and files inspected

The repository is hosted on GitHub [5] and includes systemd packaging plus a fairly elaborate execution subsystem. [6]

Files most relevant to your symptom

File	Key functions / classes (by inspection)	What it likely contributes to the "works after restart, fails later" symptom
`packaging/systemd/ai-trading.service`	systemd unit wiring (ExecStart / env / restart)	Determines how env/config is injected and how logs reach the journal; also determines whether restarts reset state often enough to "mask" the underlying issue.
`packaging/systemd/ai-trading-api.service`	separate service unit (API side)	Can be a second process that shares config/env and may affect rate-limits or provider state if it calls the same broker/data endpoints.
`ai_trading/__main__.py`	CLI entrypoint + loop driver	Controls how the main loop runs and what exceptions are treated as "recoverable" vs fatal; can inadvertently swallow relevant failures if not logged clearly.
`ai_trading/main.py`	`main()`, cycle scheduler/warmup behavior	Implements the "startup behavior vs loop behavior" split (warmup cycle then steady-state cycles), consistent with your report of "immediate after restart" vs "once in main loop." [1]
`ai_trading/execution/live_trading.py`	`ExecutionEngine` (live submission), `execute_order`, `submit_market_order`, retry/circuit/lockouts, many preflight gates	Contains multiple *stateful* and *time-dependent* "skip/suppress" mechanisms (broker lockouts, safe mode, quote gating, duplicate intent window, order pacing/caps, PDT/capacity checks) that can stop later submissions while allowing an initial submit right after restart. [2]

File	Key functions / classes (by inspection)	What it likely contributes to the "works after restart, fails later" symptom
`ai_trading/execution/engine.py`	"institutional" engine + `OrderManager`	Contains internal tracking/ idempotency/monitoring; depending on wiring, could prevent "duplicate" submits or keep a local view inconsistent with broker reality.
`ai_trading/monitoring/order_health_monitor.py`	order health tracking	Helps detect "orders not progressing / stale" vs "orders never submitted"; useful for distinguishing failure class.
`ai_trading/tools/submit_order_once.py`	single-shot submission tool (not fully retrievable here)	Strong candidate for a "startup smoke submit" (minimal gates) vs "main loop submit" (full gates). The file exists, but content retrieval was blocked in this environment, so code-path comparison is limited.

Repo file links (commit-pinned)

To keep links stable, these are pinned to the commit SHA shown by the connector results:

```
packaging/systemd/ai-trading.service
https://github.com/dmorazzini23/ai-trading-bot/blob/
7fd45b9b3be83872278b27cd1807fa1166b449e6/packaging/systemd/ai-trading.service

packaging/systemd/ai-trading-api.service
https://github.com/dmorazzini23/ai-trading-bot/blob/
7fd45b9b3be83872278b27cd1807fa1166b449e6/packaging/systemd/ai-trading-
api.service

ai_trading/__main__.py
https://github.com/dmorazzini23/ai-trading-bot/blob/
7fd45b9b3be83872278b27cd1807fa1166b449e6/ai_trading/__main__.py

ai_trading/main.py
https://github.com/dmorazzini23/ai-trading-bot/blob/
7fd45b9b3be83872278b27cd1807fa1166b449e6/ai_trading/main.py

ai_trading/execution/live_trading.py
https://github.com/dmorazzini23/ai-trading-bot/blob/
```

```
7fd45b9b3be83872278b27cd1807fa1166b449e6/ai_trading/execution/live_trading.py

ai_trading/execution/engine.py
https://github.com/dmorazzini23/ai-trading-bot/blob/
7fd45b9b3be83872278b27cd1807fa1166b449e6/ai_trading/execution/engine.py
```

Execution path from systemd start to main loop submissions

What systemd is responsible for

Under systemd, your service's environment and IO routing depend primarily on the unit file and `systemd.exec` defaults:

- `Environment=` / `EnvironmentFile=` define environment variables for the service process. [7]
- systemd generally connects stdout/stderr to the journal by default (unless overridden), and journald splits stream logs by newline/NUL. [8]
- systemd's restart policy (`Restart=...`) determines whether the process will be brought back after failure; for long-running services, `Restart=on-failure` is commonly recommended. [9]
- systemd documentation explicitly warns that **environment variables are not suitable for secrets**; consider credentials features if you're passing keys through env. [7]

These details matter because an observed "it works after restart" pattern can be caused by restarts resetting in-memory execution state, not by fixing an underlying broker/config problem.

High-level flowchart of runtime behavior

This flowchart is deliberately focused on the "first submit after restart" vs "later submits" divergence.

```
flowchart TD
  A[systemd starts ai-trading.service] --> B[ExecStart: python -m ai_trading]
  B --> C[ai_trading/__main__.py: CLI + loop driver]
  C --> D[ai_trading/main.py: main()]
  D --> E[Initialize runtime/config + HTTP session]
  E --> F[Warmup cycle (startup-only)]
  F --> G[run_cycle(): strategy -> order intents]
  G --> H[ExecutionEngine.execute_order() in execution/live_trading.py]
  H --> I{Preflight gates?}
  I -->|Pass| J[Submit via Alpaca TradingClient.submit_order()]
  I -->|Fail/Skip| K[Return None / suppress submission, log reason (sometimes
only at warn/info)]
  J --> L[Order ack/poll/reconcile; update runtime_state]
  K --> M[Cycle ends, scheduler sleeps]
  L --> M
  M --> N[Main loop: repeat run_cycle()]
  N --> H
  H --> O{Stateful conditions accumulate?}
```

```
O -->|Yes: broker lock/safe mode/quote gate/caps/duplicates| K
O -->|No| J
```

The important implication is that if the post-restart submit happens in the warmup cycle (or before stateful suppressors trip), you will see “success” right after restart, followed by “no submits” in steady state.

Code-path differences that can explain “first submit works, later ones do not”

Because I could inspect `ai_trading/main.py` and `ai_trading/execution/live_trading.py` but could not retrieve `ai_trading/tools/submit_order_once.py` content in this environment, I can only partially compare “first run vs loop” behavior. Still, the existing codebase shows two strong divergence patterns:

Startup and warmup behavior vs steady-state

`ai_trading/main.py` is explicitly structured to run a one-time startup behavior (warmup) and then enter the repeated execution loop. That alone can explain why “orders submit immediately after restart” (in warmup) but later fail (steady state), especially when later cycles hit quote/provider gates or per-cycle caps. [1]

Practical diagnostic: in journald, compare logs immediately after `systemctl restart` to logs after the first sleep interval—do you see different tokens (e.g., “warmup”/“startup reconcile” vs “price gated”/“broker suppressed”)?

Order submission in `execution/live_trading.py` can “fail” without exceptions

The live execution class contains many *intentional* early-return paths that yield “no submit” without raising:

- safe-mode / provider-disabled / halt-file blocking
- quote-quality gating (bid/ask freshness or NBBO requirements)
- broker lock suppression after an “unauthorized” condition
- PDT restrictions, capacity checks, opposite-side conflict policies
- duplicate intent suppression windows, per-cycle pacing caps

All of these can cause “no order was submitted” even though upstream logic believes it tried.

At the broker level, actual network/API failures are also handled, but often converted into retries, suppression windows, or `None` outcomes rather than hard crashes. [2]

Hypotheses for failure modes and how to confirm each in logs

This section prioritizes hypotheses that (a) match your symptom pattern and (b) are strongly suggested by the repository’s execution code.

Broker auth issues leading to timed submission suppression

Hypothesis: the bot hits a broker auth/permission error after the first successful submit (e.g., 401/403 or an SDK-raised API error). The execution engine then locks broker submissions for a cooldown window and later iterations suppress submits.

Why it fits the symptom: restart clears the lock state; you get 1–N successful submits; then an auth-related failure locks submissions again.

How to confirm: in `journalctl`, search for terms consistent with this repository's logging (examples): `BROKER_UNAUTHORIZED`, `BROKER_SUBMIT_SUPPRESSED`, `EXECUTION_CREDS_UNAVAILABLE`, `ALPACA_ORDER_SUBMIT_ERROR`, or a `status_code` around 401/403. [2]

Root causes to consider: - wrong base URL vs key type (paper vs live mismatch) - keys rotated/revoked while process is running - credentials not actually present in the steady-state environment (e.g., overwritten config reload, different environment between a one-shot tool and the daemon process) - time-dependent permission changes on the account (less common)

Provider safe-mode / disablement blocks submission after data degradations

Hypothesis: something in the runtime sets a safe-mode flag or disables the primary provider ("alpaca") after it detects degraded data or broker connectivity, and that block persists in steady state.

Why it fits the symptom: right after restart, provider monitor state is "clean"; after some degraded conditions accumulate (stale quotes, repeated broker errors), it disables the provider and submission is blocked.

How to confirm: search logs for `ORDER_BLOCKED_SAFE_MODE`, `SAFE_MODE_*`, `provider disabled`, or `QUOTE_QUALITY_BLOCKED` / `ORDER_SKIPPED_PRICE_GATED`. [2]

Quote-quality gating prevents orders once quotes become stale

Hypothesis: the execution engine requires bid/ask quotes (and possibly real-time NBBO) for limit/entry orders. After some time, quotes become stale or are sourced from a "fallback" provider, causing the gate to block new orders.

Why it fits: immediately after restart, quotes are fresh; later, if streaming/market-data refresh fails, quote ages exceed thresholds and orders stop.

How to confirm: grep journald for tokens like `ORDER_SKIPPED_PRICE_GATED`, `ENTRY_BLOCKED_BY_QUOTE_QUALITY`, or logs that include `quote_age_ms`, `provider`, `nbbo_required`, `degraded`. [2]

Per-cycle order pacing cap / duplicate intent suppression

Hypothesis: the loop generates multiple order intents for the same symbol/side or exceeds a per-cycle cap, causing later submits to be skipped (not errored).

Why it fits: after restart, the counters are reset; later in the loop, repeated intents get suppressed.

How to confirm: search for `ORDER_PACING_CAP_HIT` and `ORDER_INTENT_SUPPRESSED_DUPLICATE`. [2]

PDT / day-trade lockout behavior

Hypothesis: after one or a few orders, the account hits a pattern-day-trader day-trade limit, triggering "skip openings" behavior.

Why it fits: you might see submits immediately after restart (if you had remaining day-trades), then later orders are blocked as counters update.

How to confirm: search logs for `PDT_LOCKOUT_ACTIVE`, `PDT_LIMIT_EXCEEDED_SWING_MODE_ACTIVATED`, or `ORDER_SKIPPED_NONRETRYABLE` with PDT-related context. [2]

Broker/API rate limiting (429) emerging over time

Hypothesis: in steady state, the bot (and possibly the API service) produces enough broker requests to exceed throttling; later submits fail with 429 or are retried poorly, leading to "no effective submits."

Why it fits: rate-limit failures are time/volume dependent and can be "fine right after restart."

How to confirm: search the journal for `429`, `Too Many Requests`, `rate limit`, or SDK retry logs. Alpaca documents throttling of API calls and 429 behavior. [4]

Important note: Alpaca's `TradingClient.submit_order()` API is straightforward, but your repository wraps it with significant gating and retry logic, so 429 may appear as "retries exhausted" rather than a raw exception. [10]

Prioritized fixes and targeted patches

This section focuses on changes most likely to reduce ambiguity and make the root cause obvious quickly.

Add a single "order attempt outcome" log with a normalized reason

Right now, the live engine already logs many events, but they are dispersed across dozens of early returns. A practical improvement is to ensure every order intent results in exactly one of:

- `ORDER_SUBMIT_ATTEMPT` (with context)

- `ORDER_SUBMITTED` (with broker IDs)
- `ORDER_SUBMIT_SKIPPED` (with a specific reason code)
- `ORDER_SUBMIT_FAILED` (with exception + status code)

Patch sketch (illustrative diff): add a small helper in `ai_trading/execution/live_trading.py` and use it for early returns.

```
diff --git a/ai_trading/execution/live_trading.py b/ai_trading/execution/
live_trading.py
@@
+def _skip_submit(reason: str, *, symbol: str | None = None, side: str | None =
None,
+                 qty: int | None = None, extra: dict[str, Any] | None = None) -
> None:
+    payload: dict[str, Any] = {"reason": reason}
+    if symbol: payload["symbol"] = symbol
+    if side: payload["side"] = side
+    if qty is not None: payload["qty"] = qty
+    if extra: payload.update({k: v for k, v in extra.items() if v is not None})
+    logger.warning("ORDER_SUBMIT_SKIPPED", extra=payload)
@@ def submit_market_order(...):
-        if _safe_mode_guard(symbol, side, quantity):
-            self.stats.setdefault("skipped_orders", 0)
-            self.stats["skipped_orders"] += 1
-            return None
+        if _safe_mode_guard(symbol, side, quantity):
+            self.stats.setdefault("skipped_orders", 0)
+            self.stats["skipped_orders"] += 1
+            _skip_submit("safe_mode", symbol=symbol, side=str(side).lower(),
qty=quantity)
+            return None
@@
-        if self._broker_lock_suppressed(symbol=symbol, side=side_lower,
order_type="market"):
-            return None
+        if self._broker_lock_suppressed(symbol=symbol, side=side_lower,
order_type="market"):
+            _skip_submit("broker_locked", symbol=symbol, side=side_lower,
qty=quantity,
+                         extra={"order_type": "market"})
+            return None
```

This does not change trading behavior; it only ensures the “why” is always visible in logs.

Surface broker lockout state and remaining cooldown in every cycle summary

Because the execution layer uses cooldown suppression after unauthorized/lock states, it's valuable to emit a per-cycle summary log from the loop driver showing:

- `broker_locked_until` (monotonic deadline)
- `broker_lock_reason`
- submitted/skipped counts by reason (per cycle)

You can do this either in `ai_trading/main.py` (cycle end) or in the execution engine's `end_cycle()`.

Make rate limiting explicit and cross-process

If both `ai-trading.service` and `ai-trading-api.service` call the broker, you can hit account-level limits faster. Alpaca describes throttling and 429 responses. [4]

Recommended fix pattern:

- Implement a shared (process-wide) rate limiter for trading endpoints.
- Ensure order submission honors `Retry-After` when present, or a conservative exponential backoff on 429.
- Reduce unnecessary polling (e.g., order-status polls) once a broker "ack" is received.

Harden systemd logging and environment correctness

To avoid "it's failing but I can't see why," ensure logs are reliably in journald:

- systemd can route stdout/stderr to the journal (`StandardOutput=` / `StandardError=`), and journald consumes stream output. [8]
- Add `PYTHONUNBUFFERED=1` so Python flushes logs promptly (especially if any logging still goes to stdout).
- Consider `PYTHONFAULTHANDLER=1` to capture hung-thread traces in crashes.
- Prefer systemd credential mechanisms over env vars for secrets; systemd warns env vars are not suitable for secrets. [7]

Example systemd override snippet (drop-in):

```
# /etc/systemd/system/ai-trading.service.d/override.conf
[Service]
Environment=PYTHONUNBUFFERED=1
Environment=PYTHONFAULTHANDLER=1
StandardOutput=journal
StandardError=journal
# Optional: tag logs
SyslogIdentifier=ai-trading
```

`StandardOutput=journal` semantics are described in systemd.exec documentation. [11]

Ensure restart policy is appropriate for a daemon

If your bot gets into an unrecoverable stuck state (deadlock, infinite retry loop, or permanent suppressor), a restart policy can recover it.

`Restart=on-failure` is commonly recommended for long-running services. [9]

This is not a substitute for fixing the root cause, but it reduces impact.

Reproduction and debugging playbook

What's missing / unspecified (limits certainty)

To fully diagnose, these details are required but not provided:

- Actual server OS/distro and systemd version (affects logging directives and credential features). [11]
- Exact `ai-trading.service` as installed on the host (it may differ from `packaging/systemd/ai-trading.service`).
- Whether you run in paper vs live (`EXECUTION_MODE`, base URL, etc.) and how credentials are supplied.
- The exchange/broker account state (PDT flags, buying power, daytrade_count/daytrade_limit).
- Representative logs around the first successful submit and the first later failure (with timestamps).

Step-by-step: confirm whether it's "skip" vs "error"

All commands below are safe to run without printing secrets.

1) Inspect the live unit and its environment wiring:

```
systemctl cat ai-trading.service
systemctl show ai-trading.service -p ExecStart -p WorkingDirectory -p
Environment -p EnvironmentFiles
```

`EnvironmentFile=` semantics are defined in systemd.exec. [7]

2) Stream logs with timestamps across the restart boundary:

```
sudo journalctl -u ai-trading.service -o short-iso -f
# In another terminal:
sudo systemctl restart ai-trading.service
```

3) Immediately after restart, capture the “success” window logs and the later “failure” window logs, then grep for these high-signal tokens (based on repo logging patterns):

```
sudo journalctl -u ai-trading.service -S "2026-02-19 00:00" -o short-iso > /tmp/
ai-trading.log

rg -n "ALPACA_ORDER_SUBMIT_ATTEMPT|ALPACA_ORDER_SUBMITTED|
ORDER_SUBMIT_RETRIES_EXHAUSTED|BROKER_UNAUTHORIZED|BROKER_SUBMIT_SUPPRESSED|
ORDER_BLOCKED_SAFE_MODE|ORDER_SKIPPED_PRICE_GATED|QUOTE_QUALITY_BLOCKED|
ORDER_INTENT_SUPPRESSED_DUPLICATE|ORDER_PACING_CAP_HIT|PDT_LOCKOUT_ACTIVE|
ORDER_SKIPPED_NONRETRYABLE" /tmp/ai-trading.log
```

Interpretation guide:

- If you see `ORDER_BLOCKED_SAFE_MODE` → focus on provider monitor / halt flag / safe-mode conditions.
- If you see `ORDER_SKIPPED_PRICE_GATED` / `QUOTE_QUALITY_BLOCKED` with quote age/provider fields → focus on market-data freshness and degraded-feed settings.
- If you see `BROKER_UNAUTHORIZED` or repeated 401/403 → focus on credentials/base URL and broker lockouts.
- If you see `ORDER_PACING_CAP_HIT` or `ORDER_INTENT_SUPPRESSED_DUPLICATE` → you’re rate-limiting yourself at the strategy layer.
- If you see lots of 429/Too Many Requests → reduce broker calls; Alpaca documents throttling and 429. [4]

Force and reproduce each failure mode (so you can confirm within minutes)

These “controlled experiments” help you map symptom → mechanism.

- **Broker lock suppression:** temporarily configure the service with an invalid key (in a safe test environment) to see if you get `BROKER_UNAUTHORIZED` and subsequent `BROKER_SUBMIT_SUPPRESSED`. Then restore.
- **Safe-mode gate:** set `AI_TRADING_HALT=1` (or create the halt file if that’s configured) and confirm `ORDER_BLOCKED_SAFE_MODE` appears; then remove.
- **Quote gating:** disconnect/disable your quote feed (or artificially set quote timestamps stale in whichever component generates quotes) and confirm `ORDER_SKIPPED_PRICE_GATED`.
- **Duplicate intent suppression:** set a small `AI_TRADING_DUPLICATE_INTENT_WINDOW_SEC` and submit two identical intents; confirm `ORDER_INTENT_SUPPRESSED_DUPLICATE`.
- **Per-cycle pacing cap:** set `EXECUTION_MAX_NEW_ORDERS_PER_CYCLE=1` and force multiple order intents in one cycle; confirm `ORDER_PACING_CAP_HIT`.
- **PDT lockout:** if you’re near the PDT limit, force one more day-trade and observe whether subsequent openings are blocked with PDT logs.

Validate the broker API call shape independently (only if needed)

If you suspect the issue is the broker SDK call itself (not gating), Alpaca's `TradingClient.submit_order()` interface is documented and should succeed with valid credentials and request objects. [10]

Because the repository wraps submission heavily, always confirm whether the SDK call is being reached (look for logs like `ALPACA_ORDER_SUBMIT_ATTEMPT` in your runtime).

Minimal "fix-first" checklist

If you want the fastest path to resolution without overhauling architecture:

1) Implement the unified `ORDER_SUBMIT_SKIPPED` log (patch above) so every non-submit is explained. 2) Confirm whether failure is **broker lock** vs **quote/safe-mode gating** vs **rate limiting** (using grep workflow). 3) Fix the triggering condition: - broker lock → credentials / base URL / account permissions - quote gating → data feed freshness / degraded-feed configuration - safe mode → provider monitor or halt flag logic - pacing/duplicates → strategy throttling logic - 429 → reduce requests and add rate limiting/backoff (Alpaca throttling documented). [4] 4) Add a per-cycle KPI log summarizing attempted/submitted/skipped counts and the broker lock state (so this never regresses silently).

---

[1] ai_trading/main.py
https://github.com/dmorazzini23/ai-trading-bot/blob/7fd45b9b3be83872278b27cd1807fa1166b449e6/ai_trading/main.py

[2] ai_trading/execution/live_trading.py
https://github.com/dmorazzini23/ai-trading-bot/blob/7fd45b9b3be83872278b27cd1807fa1166b449e6/ai_trading/execution/live_trading.py

[3] [6] https://github.com/dmorazzini23/ai-trading-bot
https://github.com/dmorazzini23/ai-trading-bot

[4] Alpaca Support - Is there a usage limit for the number of API calls per second?
https://alpaca.markets/support/usage-limit-api-calls?utm_source=chatgpt.com

[5] packaging/systemd/ai-trading-api.service
https://github.com/dmorazzini23/ai-trading-bot/blob/7fd45b9b3be83872278b27cd1807fa1166b449e6/packaging/systemd/ai-trading-api.service

[7] systemd.exec
https://www.freedesktop.org/software/systemd/man/systemd.exec.html?utm_source=chatgpt.com

[8] [11] systemd.exec - CS50 Manual Pages
https://manual.cs50.io/5/systemd.exec?utm_source=chatgpt.com

[9] systemd.service
https://www.freedesktop.org/software/systemd/man/latest/systemd.service.html?utm_source=chatgpt.com

[10] Orders - Alpaca-py
https://alpaca.markets/sdks/python/api_reference/trading/orders.html?utm_source=chatgpt.com