Online Appendix 1: GLEES Implementation Details

This online appendix documents the implementation of the GLEES (GPT-Language-based Experimental Economics System) framework (Chen & Wu, 2025) for the attention-driven retail trading experiment. The implementation follows GLEES's four-component prompt architecture and extends the framework to a dynamic, multi-period trading environment with staggered treatment assignment.

1. Software Architecture

The experiment is implemented in R (version 4.5.2) using the following core packages:

Package	Purpose
R6	Object-oriented Agent and Game classes
data.table	High-performance data manipulation
jsonlite	JSON parsing for LLM responses
httr2	OpenAI Chat Completions API communication
logger	Audit trail generation
did	Callaway-Sant'Anna estimator
fixest	Two-way fixed effects estimation
future / future.apply	Parallel processing for agent decisions
cachem / memoise	Disk-based LLM response caching
digest	Cache key generation
pwr	A priori power analysis
ggplot2 / gridExtra	Visualization
modelsummary / broom	Regression output tables

2. GLEES Four-Component Prompt Architecture

Following Chen and Wu (2025), each agent interaction consists of four concatenated prompt components:

2.1 Component 1: Persona (System Prompt)

Defines stable investor characteristics assigned at initialization. Six distinct personas are implemented:

> SocialMomentum: "You are a Robinhood user who trades based on social signals and crowd behavior. You pay close attention to what other retail traders are buying. You experience FOMO."
>
> LotterySeeker: "You treat trading like buying lottery tickets; you chase volatile meme stocks like AMC and GME."
>
> PassiveFollower: "You prefer familiar large-cap names and hold positions rather than trade frequently."
>
> SkepticalContrarian: "You fade social media hype and avoid viral stocks, preferring moderate attention names."
>
> TechnicalTrader: "You are an active trader focused on price momentum and chart patterns."
>
> DividendSeeker: "You only trade stable blue-chip companies like AAPL and NVDA; you avoid AMC and GME."

All personas are passed to the OpenAI Chat Completions API as system messages and carry a sensitivity level (high/medium/low) that influences persona-specific behavior modifications.

2.2 Component 2: Game Instructions (User Prompt Section 1)

Provides invariant rules of the trading environment, constructed at experiment initialization. The implementation supports multiple attention specification modes:

- **Original:** Asymmetric constraint—agents must research (DEEP or QUICK) to buy, but can sell any position

- **Specification A:** Symmetric constraint—attention required for both buying AND selling
- **Specification B:** No restrictions—attention allocation tracked but does not constrain trading
- **Specification C:** Search cost model—ignoring research increases transaction costs (50 bps penalty for ignore, 10 bps for quick, 0 for deep)

2.3 Component 3: History (User Prompt Section 2)

Implements salience-weighted episodic memory following GLEES specifications with a recency weighting parameter of λ=0.3 and payoff-magnitude weighting. The history writer function (salience_history_writer) retrieves K=3 most recent salient events with performance indicators:

" 💰 Portfolio: $10,488 (+4.9%). Recent: [t=60] buy AMC → -$310.00 | [t=61] hold → $0.00 | [t=62] sell AMC → +$22.50"

2.4 Component 4: Stage Instructions (User Prompt Section 3)

Dynamically generated for each of three stages per period via dedicated R6 Stage classes:

Stage 1: Attention Allocation (StageAttentionAllocation)

Agents receive a social leaderboard showing stock activity levels with emoji indicators (🏆 for leader, 🔥 for viral). The prompt requires JSON output with a manipulation check:

```
{
 "manipulation_check": {
  "reasoning": "I see [TICKER] has the 🏆 flag...",
  "highest_buzz_stock": "TICKER",
  "confidence": 10
 },
 "allocation": {"AAPL": "deep/quick/ignore", ...}
}
```

Stage 2: Information Revelation (StageInformationRevelation)

Based on attention allocation, constructs individualized information panels with momentum signals, fundamental data, and news sentiment. Deep research includes full signal strength classification (noise/weak/medium/strong).

Stage 3: Trading Decision (StageTradingDecision)

Agents receive their portfolio state, researched stock signals, and persona-specific guidance reinforcement. The trading decision uses a more capable model ("System 2" thinking):

```
{"action": "buy"|"sell"|"hold", "ticker": "AAPL"|"NVDA"|"AMC"|"GME"|null,
 "shares": integer, "rationale": string}
```

3. LLM Configuration and Dual-Model Architecture

The experiment implements a dual-model "System 1 / System 2" architecture using OpenAI's Chat Completions API:

```
CONFIG$llm <- list(
  model         = "gpt-4o-mini",      # Fast model (attention/info stages)
  trading_model  = "gpt-4.1-mini",     # Slow model (trading decisions)
  endpoint      = "chat/completions",
  max_output_tokens = 2048,
  temperature    = 0.4,
```

```
  use_json_mode  = TRUE            # Enforces structured JSON output
)
```

The LLMClient R6 class (lines 717-823) manages API communication with built-in retry logic, disk caching, and automatic JSON mode routing for prompts expecting structured output. All API calls are logged with timestamp, full prompt, raw response, token usage, and latency.

4. Agent and Game Engine Implementation

Agents are implemented as R6 objects (Agent class) maintaining persistent state including cash, portfolio positions, attention budget, and episodic memory. The Game class orchestrates multi-period execution:

```
Game$new(
  name    = "AttentionShockTrading",
  rules   = game_rules,
  stages  = list(
    StageAttentionAllocation$new(),
    StageInformationRevelation$new(),
    StageTradingDecision$new()
  ),
  agents   = agents,          # 96 agent instances
  matching = matching_func,   # Treatment-aware context generator
  periods  = 252,
  seed     = 42
)
```

5. Experimental Design and Treatment Assignment

The experiment uses a staggered difference-in-differences design with three cohorts:

- **Cohort 1:** 32 agents, treatment begins at t = 60
- **Cohort 2:** 32 agents, treatment begins at t = 120
- **Cohort 3:** 32 agents, never treated (pure control)

Assignment is balanced across personas using stratified randomization, ensuring each cohort contains approximately equal representation of all six investor types. The treatment assignment is generated via create_balanced_assignment() (lines 2730-2790).

6. Price Path Simulation

Asset prices follow geometric Brownian motion with calibrated volatilities:

Ticker	Initial Price	Daily Vol	Annualized Vol
AAPL	$150	2.5%	~40%
NVDA	$450	4.5%	~71%
AMC	$12	8.0%	~127%
GME	$25	9.0%	~143%

A common drift of 0.03% per period is applied across all assets. The simulate_price_paths() function generates these paths with regime-switching volatility capability.

7. Attention Shock Treatment

The treatment consists of a viral attention shock to meme stocks (AMC, GME). Treatment and control parameters:

Metric	Control (Pre-treatment)	Treatment (Post-shock)
Mentions	150-500	45,000-60,000
Robinhood Rank	45-67	1
Retail Buyers	380-450	90,000-120,000
Buzz Level	"low"	"viral"

This represents a 90-400x increase in social metrics for treated agents. Blue-chip stocks (AAPL, NVDA) maintain moderate baseline attention (800-1,200 mentions, rank 8-15) throughout, providing within-period controls. The implementation also supports dose-response experiments with configurable intensity levels.

8. Manipulation Checks

Embedded in Stage 1 attention allocation via the manipulation_check field in the JSON response. Agents must identify the highest-buzz stock and confidence level. The actual_highest_buzz is computed from social signal data, and manipulation_check_passed is logged as a binary indicator.

9. JSON Parsing and Repair Mechanisms

The implementation includes robust JSON handling with automatic repair:

1. **clean_json_response():** Strips markdown code fences and whitespace
2. **extract_first_json_object():** Extracts balanced JSON object using brace counting
3. **Repair fallback:** On parse failure, a repair prompt is sent to the LLM with context to regenerate valid JSON
4. **Persona-specific defaults:** If repair fails, persona-appropriate fallback allocations are applied

All parse attempts are logged with json_fallback_used and parse_repaired indicators for subsequent sensitivity analysis.

10. Momentum and Signal Calculation

Price momentum is calculated with Bayesian shrinkage to handle early-period data sparsity:

```
calculate_momentum_with_shrinkage <- function(price_history, volatility_prior, target_periods = 5) {
  shrinkage_weight <- available_periods / (available_periods + target_periods)
  shrunk_momentum <- shrinkage_weight * raw_momentum_per_period
}
```

Signal strength is classified into four categories based on z-score thresholds: noise ($|z| < 0.10$), weak ($0.10 \leq |z| < 0.25$), medium ($0.25 \leq |z| < 0.50$), and strong ($|z| \geq 0.50$).

11. Data Collection and Audit Trail

All experimental events are logged to a data.table with comprehensive fields including:

- Period (t), agent_id, stage, specification
- Treatment cohort, is_treated_now, post_treatment indicator
- Attention allocation for each ticker
- Trading action, ticker, shares, price, transaction costs
- Cash, portfolio value, realized P&L
- Manipulation check results
- JSON fallback and intervention flags
- Full rationale text from LLM

Atomic save operations (write_atomic_csv, save_atomic_rds) ensure data integrity. Output files include experiment_results_final.csv, experiment_results_clean.csv (excluding intervention-affected observations), and experiment_full_log.csv.

12. Statistical Analysis Pipeline

The prepare_clean_sample() function creates intervention-clean datasets. Two-way fixed effects estimation uses fixest:

```
model <- fixest::feols(buy_indicator ~ is_treated_now | agent_id + t,
                       data = clean_trading, cluster = "agent_id")
```

Callaway-Sant'Anna difference-in-differences estimation is available via the did package for event study analysis and dynamic treatment effect estimation.

13. Reproducibility and Replication Package

Complete replication materials include:

- Attention_driven_trading__final_.R (3,605 lines, complete implementation)
- treatment_assignment.csv (96 agents with cohort assignments)
- experiment_results_final.csv / .rds (full experimental log)
- experiment_metadata_corrected.rds (configuration and feature flags)
- experiment_audit.log (API call log with timestamps)
- power_summary.csv (a priori power analysis results)

Exact replication requires:

- R version 4.3.0 or higher
- OpenAI API key with GPT-4o-mini and GPT-4.1-mini access
- Total observations: 96 agents × 252 periods × 3 stages = 72,576
- Estimated API cost: ~$8-12 USD at current rates

14. Comparison to GLEES Validation Studies

This implementation extends Chen and Wu (2025) in several dimensions:

Feature	GLEES Validation	This Study
Game Type	Static (ultimatum, trust)	Dynamic portfolio choice
Action Space	Discrete	Continuous (shares)
Periods	1-20 rounds	252 periods
Treatment	Between-subject	Staggered within-subject
Memory	Simple history	Salience-weighted retrieval
LLM Architecture	Single model	Dual-model (System 1/2)

Key technical innovations:

1. **Multi-stage prompting:** Three distinct prompt constructions per period vs. single-stage in GLEES
2. **Dynamic budget constraints:** Attention points and cash constraints create realistic scarcity
3. **Treatment manipulation:** 90-400x algorithmic ranking intervention vs. persona-only manipulations
4. **Path dependence:** Portfolio positions carry forward creating history-dependent contexts
5. **Dual-model architecture:** Different models for automatic vs. deliberative decisions

15. Limitations and Extensions

Several GLEES features could not be fully implemented:

- **Agent-agent interactions:** Current implementation uses exogenous GBM price paths. Future extensions could implement order book mechanisms where trades affect prices.

- **Adaptive learning:** While agents maintain memory, they do not explicitly update belief models. Reinforcement learning extensions could allow strategy optimization.
- **Natural language interaction:** Agents make structured JSON decisions. Discussion forums or negotiation could be implemented per GLEES specifications.

References

Chen, K.-Y., & Wu, Y. D. (2025). GPT-Language-based Experimental Economics System (GLEES). *Working Paper*, University of Texas at Arlington and San Jose State University.

Callaway, B., & Sant'Anna, P. H. (2021). Difference-in-differences with multiple time periods. *Journal of Econometrics*, 225(2), 200-230.

OpenAI. (2024). *OpenAI API Documentation: Chat Completions*. https://platform.openai.com/docs/api-reference/chat