Fernwey × Prediction Markets (Polymarket + Kalshi) Stock-Market Sentiment & Event-Impact Module — Work Plan

Goal: turn prediction-market prices (implied probabilities) into a clean, comparable, explainable sentiment + event-risk signal for equities, and integrate it into Fernwey's research/alerts workflow.

1) What the paper contributes (in plain English)

· Prediction markets (e.g., Polymarket) often create multiple related contracts about the same underlying event; contract prices should be internally consistent but sometimes aren't, creating arbitrage opportunities.

· The paper proposes a scalable way to find and categorize dependency relationships between markets/conditions (e.g., 'Team A wins' implies 'Team A wins by ≥2'), so you can compare correlated markets efficiently.

· They use (a) text embeddings to cluster/triage related markets by topical similarity + time proximity, and (b) LLMs to extract logical relationships from market descriptions in a structured format (e.g., JSON).

· They then use historical order book / trade data to measure when inconsistencies existed and whether they were exploited (two types: intra-market rebalancing vs inter-market combinatorial arbitrage).

· Takeaway for Fernwey: the key engineering is the dependency graph + probability aggregation layer that turns many messy contracts into one clean probability per event.

2) Why this matters for stock analysis (Fernwey use case)

· Prediction markets are real-money, high-frequency 'belief aggregators'—often faster than news or social media for event probabilities.

· For equities, the useful output is not 'arbitrage'; it's an event probability curve over time (and confidence/uncertainty), which can be converted into a backtestable sentiment/risk factor.

· Examples: elections affecting sectors, regulatory rulings affecting a company (antitrust, approvals), macro prints (CPI, rate decisions), litigation outcomes, M&A approvals, geopolitics.

· Fernwey can surface: (1) market-implied probability of event outcomes, (2) probability shock (how quickly it changed), and (3) dispersion (disagreement across markets/platforms).

3) Product concept: 'Event Probability → Equity Sentiment' pipeline

Core idea: ingest Polymarket + Kalshi markets, clean/normalize probabilities, aggregate correlated markets into a single event probability, and map events to tickers/industries to produce equity-relevant signals.

3.1 Outputs Fernwey should expose to users

· Event Dashboard: probability timeline (P(t)), volume/liquidity, and key moments (big moves).

· Sentiment Score: standardized z-score of probability movement and momentum (e.g., 1d/7d change).

· Uncertainty/Dispersion: spread across sources (Polymarket vs Kalshi) and across correlated contracts.

· Equity Impact Lens: 'Which tickers are most exposed?' + concise explanation linking event → fundamental drivers.

· Alerts: threshold-based (e.g., probability crosses 60%), shock-based (ΔP > X% in Y hours), divergence-based (platform disagreement > X%).

4) End-to-end workflow (engineering + research)

4.1 Data ingestion

· Polymarket: market metadata (question/description, outcomes/conditions, end_date, category), price/time series, liquidity/volume; optionally on-chain events if needed.

· Kalshi: contract metadata + price/time series + volume/order-book if available.

· News/filings (Fernwey existing): for explaining why probability moved and for mapping to tickers.

4.2 Market normalization (make probabilities comparable)

· Binary markets: interpret price as implied probability (optionally adjust for fees/spread).

· Multi-outcome markets ('Negative Risk'): ensure outcomes are mutually exclusive and collectively exhaustive; normalize so $\Sigma p_i \approx 1$.

· Edge cases: duplicated markets, inconsistent end dates, thin liquidity; down-weight low-liquidity markets.

· Create a canonical ‘Event’ object: {event_id, title, end_time, platform_markets[], outcomes[], tickers[] }.

4.3 Correlated market linking (the paper’s core idea)

· Step A — Candidate generation (cheap): group markets by topic + time proximity using embeddings; filter by overlapping named entities (people/places/tickers) and end dates.

· Step B — Relationship extraction (precise): use an LLM to infer logical dependencies between contracts/outcomes (implication, complement, subset). Output a structured JSON graph.

· Step C — Human-in-the-loop: for high-impact events mapped to major tickers, allow analysts to approve/override relationships.

4.4 Probability aggregation into one ‘event probability’

· Within a single multi-outcome event: use normalized outcome probabilities (p1…pk).

· Across correlated markets: combine probabilities using a weighted ensemble (weights = f(liquidity, volume, recency, platform reliability, spread)).

· Produce: P_event(outcome=i, t) + confidence bands (e.g., via dispersion or Bayesian shrinkage).

· Derive features: ΔP (1h/24h/7d), momentum, probability volatility, and ‘surprise’ vs baseline.

4.5 Mapping events to equities (impact graph)

· Ticker/entity linking: NER + knowledge base; store {event ↔ ticker} with rationale.

· Impact type: revenue risk, margin risk, regulatory risk, cost of capital, demand shock, supply shock, sentiment-only.

· Directionality: each outcome has a sign (+/−/mixed) per ticker; start with simple rules + manual overrides.

4.6 Stock-side modeling & backtesting

· Build datasets: event-probability features aligned to equity returns/vol/volume (intraday + daily).

· Backtests: event-window studies and intraday shock windows; compare to baselines (news-only, social sentiment).

· Model options (start simple): linear/logistic models using ΔP and momentum; then add interactions (sector, liquidity, macro regime).

- Evaluation: information coefficient (IC), hit-rate around shocks, incremental explanatory power vs standard factors, stability across time.

4.7 Fernwey UX integration (web + iMessage)

- Web Hub: event pages with probability chart, related markets, and ‘What changed?’ auto-summary with sources.
- Chat/iMessage: user asks ‘How is the market pricing {event}?’ → current probability + recent move + implication for tickers.
- Alerts: subscribe to {ticker} or {event}; push probability changes + short explanation.
- Audit trail: store aggregation inputs + dependency graph snapshot used for each answer.

5) Implementation plan (phased)

Phase 0 — 1 week: scope + schemas

- Define Event schema + Market schema + Mapping schema.
- Pick 10–20 high-value events mapped to public tickers for a pilot.
- Decide minimal data: price/time series + volume + descriptions.

Phase 1 — 2–3 weeks: ingestion + normalization MVP

- Integrate APIs (Polymarket + Kalshi) into a unified data store.
- Implement normalization for binary + multi-outcome; compute Σp checks and liquidity filters.
- Build a simple event page showing P(t) for one outcome + volume.

Phase 2 — 2–4 weeks: correlation linking + aggregation

- Embedding-based candidate linking + LLM relationship extraction to produce dependency graphs.
- Weighted aggregation across correlated markets/platforms; output confidence/dispersion metrics.
- Unit tests: known synthetic examples (e.g., winner vs winner-by-margin).

Phase 3 — 3–6 weeks: equity mapping + sentiment factor

- Event↔ticker mapping pipeline with human review UI.
- Compute sentiment features and run event studies; document ‘where it adds signal.’

· Add alert triggers and chat responses.

Phase 4 — ongoing: scale + hardening

· Expand event categories; improve relationship extraction prompts; add platform-specific adjustments.

· Monitoring, data-quality checks, and drift dashboards.

· Compliance: clear disclaimers; focus on data + research tools (no personalized advice).

6) Mind map (copy/paste friendly)

```
Fernwey Prediction-Market Module
├─ Data Sources
│  ├─ Polymarket (metadata, prices, liquidity, on-chain)
│  ├─ Kalshi (metadata, prices, volume)
│  └─ News/Filings (explanations + ticker mapping)
├─ Normalization Layer
│  ├─ Binary probs (YES/NO)
│  ├─ Multi-outcome / Negative Risk normalization (Σp=1)
│  └─ Quality weighting (liquidity, spread, recency)
├─ Correlation & Dependency Graph
│  ├─ Candidate linking (embeddings + time proximity + entities)
│  ├─ LLM relationship extraction (implication/subset/complement)
│  └─ Human review for high-impact events
├─ Aggregation Engine
│  ├─ Combine correlated markets → single event probability P(t)
│  ├─ Confidence/dispersion across markets/platforms
│  └─ Features (ΔP, momentum, volatility, divergence)
├─ Equity Impact Layer
│  ├─ Event↔Ticker mapping + rationale
│  ├─ Outcome directionality (+/−/mixed)
│  └─ Outputs: sentiment factor + risk alerts
└─ Fernwey Product Surface
   ├─ Web Hub (charts, related markets, “what changed”)
   ├─ iMessage chat (Q&A + explainers)
   └─ Alerts (threshold/shock/divergence) + audit trail
```

7) Deliverables checklist

· Unified market/event database + API layer

· Normalization + Negative Risk handling

· Correlation linking + dependency graph JSON

· Aggregation outputs: P(t), dispersion, features

· Equity mapping table: event↔ticker↔directionality

· Backtest report: does it add signal?

· Fernwey UI components (event page, chat responses, alerts)