Spagbot 1.0 : An AI Ensemble Forecaster

Date: 05 September2025
Contact kevin dot wyjad at gmail dot com
Written by GPT5

Spagbot: how it works (a clear, non-coder's tour)

Below is a plain-English walkthrough of the Spagbot repository and how the system makes forecasts end-to-end.

---

1) What Spagbot is trying to do

Spagbot is a general-purpose forecasting bot, currently structured to participate in Metaculus AI forecasting tournaments. For each active Metaculus question, it:

1. gathers recent, decision-useful context ("research report");
2. asks several LLM "panelists" (ChatGPT via OpenRouter, Claude, Gemini, Grok) to produce a forecast in a strict format;
3. (optionally) runs a simple game-theoretic bargaining simulation (GTMC1) when the topic looks strategic (e.g., elections, conflict, coalitions) to produce another probability-like signal;
4. **combines all evidence with a small-n Bayesian/MCMC layer** to produce the final forecast;
5. writes clean logs/CSVs, and (if enabled) can submit the forecast back to Metaculus.

Think of it as: **research brief → multi-model panel → (strategic simulation) → Bayesian aggregator → forecast + logs.**

---

2) The main script that orchestrates everything

spagbot.py (the "conductor")

When you run python spagbot.py --mode test_questions:

- **Loads configuration and API keys** from your environment (e.g., OpenRouter key, Google key for Gemini, xAI key for Grok, Metaculus token).
- **Picks questions** (either a fixed test set or live from a Metaculus tournament).
- For each question, runs the pipeline below and saves outputs.

The pipeline (per question)

A) Research brief

Spagbot builds a compact research prompt that asks an LLM to produce a short, structured brief: reference classes (base rates), recent timeline items, drivers, differences vs. base case, a Bayesian-update sketch, indicators, and caveats. If you've configured the AskNews key, it can pull 4–8 recent items to ground the brief; otherwise it leans on general knowledge. Spagbot also

appends a **"Market Consensus Snapshot"** that extracts the current Metaculus community view and attempts a best Manifold match (useful as another weak piece of evidence).

B) Panel forecasts (LLM ensemble)

Spagbot prompts each model with a **strictly formatted** forecasting instruction tailored to the question type:

- **Binary**: the model must end with Final: ZZ%.
- **Numeric**: the model must end with six lines P10, P20, P40, P60, P80, P90.
- **MCQ**: it must output one Option_i: XX% per choice (summing ~100%).

Spagbot **parses only those last lines**, not the free-text reasoning. This protects against meandering outputs and lets the aggregator read clean numbers.

C) Optional GTMC1 strategic simulation (binary only, when relevant)

If the title contains strategic keywords (coalition, ceasefire, election, strike, sanction, etc.), Spagbot asks an LLM to extract a small **actor table** (3–8 actors with position/capability/salience/risk threshold along a 0–100 policy axis where 100 ≈ "YES" side). That table feeds **GTMC1**, a compact bargaining/coalition formation simulator (details below). GTMC1 returns a probability-like signal (the share of Monte Carlo runs where the equilibrium median is ≥50 on the axis), plus diagnostics (coalitions, dispersion, rounds).
Spagbot then **adds this as another piece of evidence** and re-prompts the panel one more time with a short GTMC1 "brief" appended to the research.

D) Bayesian / Monte-Carlo aggregator (the decider)

This is where the model-by-model outputs (and GTMC1, if present) are fused into the final forecast:

- **Binary**: treat each model's % yes as a soft count in a **Beta** prior/posterior (very weak prior so the data dominates), draw samples via Monte Carlo, and report the median as the final probability.
- **MCQ**: treat each model's probability vector as soft counts in a **Dirichlet** prior/posterior and use the posterior mean.
- **Numeric**: for each model's (P10/P50/P90), fit a **separate normal** that reproduces those quantiles, then **mix** the normals in proportion to each model's weight. Monte Carlo draws from this mixture produce the final P10/P50/P90.

This does two social-science-friendly things:

1. treats each forecaster like a small "survey" contributing fractional evidence, and
2. **keeps disagreement intact** (especially for numeric), rather than prematurely averaging.

E) Outputs (where to look)

- forecasts.csv → one row per question with the final result (binary prob, MCQ dict, numeric P10/50/90), plus summaries of GTMC1 and Bayes-MC.

- forecasts_by_model.csv → per-model parsed outputs (useful to see who failed/parsed).
- forecasts_mcq_wide.csv → MCQ results in a fixed "wide" format (labels and probs in columns).
- forecast_logs/<RUNID>_reasoning.log → a human-readable log containing:
    - the research brief (and market snapshot),
    - GTMC1 section (whether it ran; the **actor table input and the GTMC1 output JSON**),
    - Bayes-MC summary (without raw samples),
    - and the **raw panel reasoning** blocks for auditability.
- logs/spagbot_run_<timestamp>.txt → a tee'd console log.

---

3) The supporting modules (why they exist and what they do)

bayes_mc.py (Bayesian + Monte Carlo combiner)

Goal: turn a handful of panel forecasts into a coherent, uncertainty-aware final prediction.

- **Binary → Beta-Binomial**: each forecast with weight *w* contributes *w·p* to "success" counts and *w·(1−p)* to "failure" counts; posterior is Beta(α,β), sampled to get median/mean and P10–P90.
- **MCQ → Dirichlet-Multinomial**: each forecast adds fractional counts per option; posterior is Dirichlet, sampled for means and intervals.
- **Numeric → Mixture of normals**: for each model's (P10/P50/P90), fit a normal that reproduces that 10–50–90 structure, then draw from the **mixture** (one component per model, weight = model weight). This preserves disagreement and is much more robust than averaging percentiles first.

You can read this file as: "**small-n Bayes** for each question type, with **Monte Carlo** for uncertainty and intervals."

GTMC1.py (game-theoretic Monte Carlo for strategic cases)

Goal: produce a **fast, interpretable bargaining signal** from a small actor table.

- **Actor table**: 3–8 actors with 0–100 **position** on a policy axis, **capability** and **salience** (both 0–100), and a **risk/threshold** parameter that affects how readily actors challenge or accept moves.

- **Dynamics**: repeated pairwise challenges where actors' positions update toward weighted midpoints if mutual expected utility clears a (learned) threshold. There's gentle **learning** of risk and threshold, and **coalitions** form if clusters are close.
- **Monte Carlo**: each *run* jitters inputs (position/capability/salience/risk), runs the dynamics until convergence or a max number of rounds, records the **final median position**, and whether coalitions formed.
- **Signal**: across runs, compute:
    - exceedance_ge_50 → fraction of runs whose final median ≥ 50 (treat as *P(YES)* on the axis),
    - coalition_rate, median_of_final_medians, median_rounds, and dispersion (IQR).
- **Artifacts**: writes a small CSV of per-run results + a tiny JSON "meta" next to it (in gtmc_logs/), which your reasoning log links to.

This is intentionally **simple and transparent**—good for a first-pass bargaining read rather than a full structural model.

calibration_advice.txt (lightweight guidance injected into prompts)

A small text file whose contents are **prefixed to every forecasting prompt** ("CALIBRATION GUIDANCE") so models see the latest "be cautious / don't swing hard" notes while you gather real performance. When you later compute Brier/CRPS/PIT diagnostics, you (or a helper script) can update this file so the next runs reflect what was learned.

---

4) The prompts (why they look long)

Each of the three forecast prompts (binary, numeric, MCQ) is deliberately **didactic**:

- **Start with a base rate** (reference class),
- **compare this case vs. the base**,
- **evaluate evidence as likelihoods** (even qualitatively),
- **do a Bayesian update sketch**,
- **red-team** your own conclusion,
- and **end with strict output lines** that Spagbot can parse automatically.

This encourages consistent, **Bayesian-style reasoning** while keeping the parsing stable.

---

5) Configuration & environment (how to turn knobs)

- **Model toggles**: environment variables like USE_OPENROUTER, USE_GOOGLE, ENABLE_GROK determine which panelists are active.
- **Model IDs**: you can point OPENROUTER_*_ID to specific models/providers available on your Metaculus-provided OpenRouter key (e.g., openai/gpt-4o, anthropic/claude-3.7-sonnet, etc.).
- **Timeouts & weights**: modest per-model timeouts; weights are simple constants and can be tuned later if calibration says a model tends to over/under-state risk.
- **Submission**: set SUBMIT_PREDICTION=1 to post back to Metaculus; otherwise it only logs locally.
- **Cache**: LLM research briefs are cached per question (so reruns don't hit the API unless you pass --fresh-research). This enables holding research steady while testing model adjustments for performance.

---

6) Inputs and outputs (what files you'll see)

Inputs / configs

- .env (or .env.template) – API keys & toggles.
- run_bot_on_*.yaml – run configurations you may keep around for CI/automation.
- pyproject.toml / poetry.lock – Python dependencies (so others can reproduce your environment).

Core code

- spagbot.py – orchestrates the run; handles research, prompting, GTMC1, Bayes-MC, logging.
- bayes_mc.py – Bayesian/Monte-Carlo combiners for each question type.
- GTMC1.py – bargaining/coalition simulator from actor tables.
- update_calibration.py (if used later) – helper to refresh calibration_advice.txt with findings.

Artifacts (created on run)

- forecasts.csv – final forecast per question (open in Excel).
- forecasts_by_model.csv – sanity check which panelists parsed OK.
- forecasts_mcq_wide.csv – fixed-column MCQ table.
- forecast_logs/<RUNID>_reasoning.log – a readable narrative log (research + GTMC1 + panel reasoning).

- logs/spagbot_run_<timestamp>.txt – tee'd console log.
- gtmc_logs/* – per-run GTMC1 CSV + meta JSON when GTMC1 activates.

---

7) How the pieces fit together (mental model)

- Treat each LLM panelist as a **noisy expert** that must speak in a standard format.
- Treat the GTMC1 output as a **bargaining-outcome clue** (only for strategic cases), not the final word.
- Treat the Bayes-MC layer as a **statistical fuse** that:
    - starts with **weak priors** (so evidence dominates),
    - is **conservative** (fractional counts instead of full counts), and
    - **keeps uncertainty** (Monte Carlo intervals; numeric mixtures preserve disagreement).

This design matches good social-science practice: respect base rates, make explicit updates from evidence, preserve uncertainty, and keep audit trails.

---

8) What to open first after a run

1. **forecast_logs/<RUNID>_reasoning.log** – read the question header, the Research section, and the GTMC1 section (if any). You'll see exactly which actor table was used and what GTMC1 concluded.
2. **forecasts.csv** – check the final probability or percentiles.
3. **forecasts_by_model.csv** – spot if a panelist failed to parse (e.g., "timeout" or "parse_failed").
4. **gtmc_logs/*_runs.csv** – (strategic questions only) skim the distribution of final medians, coalitions, and rounds.

---

9) Common "what if?" questions

- **What if a panelist ignores the format?**
  Spagbot only trusts the strict final lines. If parsing fails, that model either gets equal-weight fallback (MCQ) or is dropped (binary/numeric). This keeps aggregation clean.
- **What if GTMC1 can't extract actors?**
  Then it simply doesn't run; the Bayesian layer proceeds with the LLM panel only.

- **Why such weak priors?**
  Early in a tournament, you don't have calibration yet. Weak priors reduce the risk of over-anchoring. Once you collect enough ground truth, you can toughen priors or reweight panelists.
- **Why a numeric "mixture" instead of averaging percentiles?**
  Averaging P10/P50/P90 squashes disagreement and breaks tails. A mixture keeps **multi-modality** and more honest uncertainty.

Spagbot as Mathematical Model

Here's the "math under the hood" for Spagbot, written to be rigorous but readable. It covers (i) how the evidence is represented, (ii) the Bayesian aggregation for each question type (binary, MCQ, numeric), (iii) the GTMC1 strategic simulation and how its signal plugs into the aggregator, and (iv) the calibration metrics and how they feed back.

---

1) Notation and high-level structure

For a single forecasting question, Spagbot turns multiple sources of evidence into a single predictive distribution:

- A panel of LLM forecasters m=1,…,Mm=1,\dots,M, each returning:
  - **Binary:** a probability pm∈[0,1]p_m \in [0,1].
  - **MCQ (K options):** a simplex vector pm=(pm1,…,pmK)\mathbf{p}_m=(p_{m1},\dots,p_{mK}) with ∑kpmk=1\sum_k p_{mk}=1.
  - **Numeric:** a set of **percentiles** {Qm(q)}q∈Q\{Q_m(q)\}_{q\in\mathcal{Q}} (e.g., q∈{0.10,0.20,0.40,0.60,0.80,0.90}q\in\{0.10,0.20,0.40,0.60,0.80,0.90\}).
- An optional **GTMC1** bargaining signal for strategic binary questions, a probability-like number pgtmc∈[0,1]p_{\text{gtmc}} \in [0,1].
- (Research + market snapshot are **context for the panel**; they are not directly combined mathematically—panelists see them in their prompts.)

Each source mm has a **weight** wm≥0w_m \ge 0 (default: equal weights). If GTMC1 runs, it has a weight wgtmc≥0w_{\text{gtmc}}\ge 0. Aggregation uses **weak priors** so that observed evidence dominates but uncertainty is preserved.

---

2) Binary questions: a Beta–Binomial style combiner

2.1 Evidence as fractional counts

Treat each forecaster's probability pmp_m as **fractional evidence** (a soft "yes" count). With prior Beta(α0,β0)\text{Beta}(\alpha_0,\beta_0) (weak, e.g., α0=β0=0.1\alpha_0=\beta_0=0.1) and weights wmw_m, the posterior hyperparameters are:

α=α0 + ∑m=1Mwm pm + wgtmc pgtmc,β=β0 + ∑m=1Mwm (1−pm) + wgtmc (1−pgtmc).\alpha = \alpha_0 \;+\; \sum_{m=1}^M w_m\, p_m \;+\; w_{\text{gtmc}}\,p_{\text{gtmc}}, \qquad \beta = \beta_0 \;+\; \sum_{m=1}^M w_m\, (1-p_m) \;+\; w_{\text{gtmc}}\, (1-p_{\text{gtmc}}).

This is the standard **conjugate update** if you think of each model contributing a soft Bernoulli "count" of size wmw_m.

2.2 Posterior summary, uncertainty, and intervals

The posterior for P(YES)P(\text{YES}) is Beta(α,β)\text{Beta}(\alpha,\beta). Useful quantities:

- **Mean:** E[P]=αα+β\mathbb{E}[P]=\dfrac{\alpha}{\alpha+\beta}.
- **Variance:** Var[P]=αβ(α+β)2(α+β+1)\mathrm{Var}[P]=\dfrac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}.
- **Quantiles / credible interval:** either use the Beta CDF inverse FBeta(α,β)−1(⋅)F^{-1}_{\text{Beta}(\alpha,\beta)}(\cdot) or sample NN draws p(s)∼Beta(α,β)p^{(s)}\sim \text{Beta}(\alpha,\beta) and take empirical quantiles (e.g., 10–90% band).

Final point estimate Spagbot reports is typically the **posterior median** (robust when priors are weak), with the full distribution used to compute intervals for logs/diagnostics.

---

3) MCQ questions: a Dirichlet combiner

3.1 Evidence as fractional category counts

Let there be KK options. Use a weak Dirichlet prior Dir(α0)\text{Dir}(\boldsymbol{\alpha}_0) (e.g., α0k=ε\alpha_{0k}=\varepsilon for all kk). Each forecaster contributes a fractional count vector wmpmw_m\mathbf{p}_m. Posterior parameters:

αk=α0k + ∑m=1Mwm pmkfor k=1,…,K,α0tot=∑k=1Kαk.\alpha_k = \alpha_{0k} \;+\; \sum_{m=1}^M w_m\, p_{mk} \quad \text{for } k=1,\dots,K, \qquad \alpha_0^{\text{tot}}=\sum_{k=1}^K \alpha_k.

3.2 Posterior summaries

The posterior distribution of the probability vector θ\boldsymbol{\theta} over options is Dir(α)\text{Dir}(\boldsymbol{\alpha}). Key summaries:

- **Posterior mean:** E[θk]=αkα0tot\mathbb{E}[\theta_k]=\dfrac{\alpha_k}{\alpha_0^{\text{tot}}}.

- **Variance:** Var(θk)=αk(α0tot−αk)(α0tot)2(α0tot+1)\mathrm{Var}(\theta_k)=\dfrac{\alpha_k\left(\alpha_0^{\text{tot}}-\alpha_k\right)}{(\alpha_0^{\text{tot}})^2(\alpha_0^{\text{tot}}+1)}.
- **Covariance:** Cov(θk,θℓ)=−αkαℓ(α0tot)2(α0tot+1)\mathrm{Cov}(\theta_k,\theta_\ell)= -\dfrac{\alpha_k\alpha_\ell}{(\alpha_0^{\text{tot}})^2(\alpha_0^{\text{tot}}+1)}.

For intervals or top-1 confidence, draw NN samples θ(s)∼Dir(α)\boldsymbol{\theta}^{(s)}\sim \text{Dir}(\boldsymbol{\alpha}) and compute empirical summaries.

---

4) Numeric questions: quantile-fitted mixture of normals

Panelists return multiple quantiles Qm(q)Q_m(q) (e.g., 10/20/40/60/80/90th). Spagbot fits **one parametric component per model**, then mixes them by weights.

4.1 Fitting a normal component to each model's quantiles

For model mm, assume a normal Xm∼N(μm,σm2)X_m\sim \mathcal{N}(\mu_m,\sigma_m^2) that approximates its reported quantiles. Let zq=Φ−1(q)z_q=\Phi^{-1}(q) be the standard normal quantile. A simple least-squares fit chooses μm,σm\mu_m,\sigma_m to minimize

∑q∈Q(Qm(q)−μm−σmzq)2.\sum_{q\in\mathcal{Q}} \left(Q_m(q) - \mu_m - \sigma_m z_q\right)^2.

Closed-form solution (linear regression of Qm(q)Q_m(q) on zqz_q):

σm=∑q(zq−z¯) (Qm(q)−Q¯m)∑q(zq−z¯)2,μm=Q¯m−σmz¯,\sigma_m = \frac{\sum_{q} (z_q-\bar z)\,(Q_m(q)-\bar Q_m)}{\sum_{q}(z_q-\bar z)^2}, \qquad \mu_m = \bar Q_m - \sigma_m \bar z,

with z¯=1|Q|∑qzq\bar z=\frac{1}{|\mathcal{Q}|}\sum_q z_q, Q¯m=1|Q|∑qQm(q)\bar Q_m=\frac{1}{|\mathcal{Q}|}\sum_q Q_m(q).
(A robust shortcut if only 10/50/90 are available is μm=Qm(0.5)\mu_m=Q_m(0.5), σm≈Qm(0.9)−Qm(0.1)2.5631\sigma_m \approx \frac{Q_m(0.9)-Q_m(0.1)}{2.5631}.)

4.2 Mixture model and summarization

Define a **mixture** over components:

f(x) = 1W∑m=1Mwm ϕ(x|μm,σm2),with W=∑mwm,f(x) \;=\; \frac{1}{W}\sum_{m=1}^M w_m \, \phi(x\mid \mu_m,\sigma_m^2), \quad \text{with } W=\sum_m w_m,

where ϕ\phi is the normal density. To report percentiles, draw NN samples: pick component mm with prob. wm/Ww_m/W, then sample x∼N(μm,σm2)x\sim\mathcal{N}(\mu_m,\sigma_m^2). The empirical percentiles Q^(q)\hat Q(q) summarize the final predictive distribution (and preserve **multi-modality** if panelists disagree).

---

5) GTMC1: a strategic bargaining Monte Carlo (binary-only add-on)

GTMC1 provides a **structural** signal for strategic yes/no questions (e.g., wars, coalitions, elections). It maps a small **actor table** into a bargaining outcome and converts that into a probability-like number. The axis is scaled so that **100 ≈ "YES"** and **0 ≈ "NO".**

5.1 Inputs per actor $i=1,\dots,n$

- Position $x_i \in [0,100]$ (policy/outcome ideal point; rescale to $[0,1]$ as $\tilde x_i = x_i/100$).
- Capability $c_i \ge 0$.
- Salience $s_i \in [0,1]$.
- Risk/threshold parameters controlling willingness to **challenge** vs. accept moves (call them ρ_i, τ_i).

Define an **influence weight** $w_i = c_i s_i$ (optionally normalized so $\sum_i w_i = 1$).

5.2 Pairwise contest model (stylized)

When actor i challenges j, a **win probability** can be modeled in Bradley–Terry style using weights and (optionally) positional distance:

$$\Pr(i \succ j) = \frac{\exp\{\lambda w_i - \gamma |x_i - x_j|\}}{\exp\{\lambda w_i - \gamma |x_i - x_j|\} + \exp\{\lambda w_j - \gamma |x_i - x_j|\}} = \frac{w_i^{\lambda}}{w_i^{\lambda} + w_j^{\lambda}} \quad (\text{if } \gamma = 0),$$

with $\lambda, \gamma \ge 0$ tuning how strongly resources and distance matter.

Actor i's **expected utility** from challenging j can be represented as

$$EU_{i \to j} = \Pr(i \succ j) \cdot U_i(\text{post-move}) - (1 - \Pr(i \succ j)) \cdot U_i(\text{counter-move}), \quad EU_{i \to j} = \Pr(i \succ j) \cdot U_i(\text{post-move}) - (1 - \Pr(i \succ j)) \cdot U_i(\text{counter-move}),$$

where U_i penalizes distance from x_i (e.g., quadratic loss). A challenge is initiated if $EU_{i \to j} > \tau_i$, i.e., it clears actor i's threshold.

5.3 Updating positions and coalition formation

If a challenge occurs, both sides adjust toward a **weighted midpoint** with **learning rate** $\eta \in (0,1]$:

$$x_i \leftarrow x_i + \eta \cdot \frac{w_j}{w_i + w_j}(x_j - x_i), \qquad x_j \leftarrow x_j + \eta \cdot \frac{w_i}{w_i + w_j}(x_i - x_j).$$

Over rounds, actors with nearby positions may **coalition-merge**: if a cluster C forms (e.g., max internal distance below a tolerance ϵ), replace it by a single actor with

$$x_C = \frac{\sum_{i \in C} w_i x_i}{\sum_{i \in C} w_i}, \qquad w_C = \sum_{i \in C} w_i,$$

and updated (ρ,τ)(\rho,\tau) (e.g., means). This models bloc formation.

5.4 Monte Carlo, signal, and diagnostics

Repeat the dynamics for RR runs with **jittered inputs** (e.g., xi,ci,si,ρi,τix_i,c_i,s_i,\rho_i,\tau_i perturbed by small zero-mean noise to reflect estimation error). For each run rr:

1. Iterate pairwise challenges until **convergence** (no profitable challenges) or a **max rounds** cap.
2. Record the **final median position** m(r)m^{(r)} (on 0–100 scale).
3. Record whether a coalition formed and how many rounds elapsed.

Signal reported to the aggregator (as an extra "panelist" on the binary scale):

pgtmc = Pr(m(r)≥50) ≈ 1R∑r=1R1{m(r)≥50}.p_{\text{gtmc}} \;=\; \Pr\big(m^{(r)} \ge 50 \big) \;\approx\; \frac{1}{R}\sum_{r=1}^R \mathbf{1}\{m^{(r)}\ge 50\}.

Diagnostics often include the **median of final medians** median{m(r)}\text{median}\{m^{(r)}\}, **coalition rate** (share of runs with coalitions), **dispersion** (e.g., IQR of m(r)m^{(r)}), and **median rounds to converge**.

This pgtmcp_{\text{gtmc}} is then inserted into the **binary Beta combiner** in §2 with weight wgtmcw_{\text{gtmc}}.

---

6) Putting it together (per question)

1. **Panel stage:** collect pmp_m / pm\mathbf{p}_m / Qm(·)Q_m(\cdot) with weights wmw_m.
2. **Optional GTMC1 (binary only):** compute pgtmcp_{\text{gtmc}} and include it with weight wgtmcw_{\text{gtmc}}.
3. **Aggregation:**
    - **Binary:** Beta(α,β)\text{Beta}(\alpha,\beta) per §2 → posterior median (and interval).
    - **MCQ:** Dir(α)\text{Dir}(\boldsymbol{\alpha}) per §3 → posterior mean vector (and intervals).
    - **Numeric:** mixture of normals per §4 → report percentiles of the mixture.
4. **Outputs:** write the final numbers and uncertainty summaries; keep all diagnostics and panel member outputs for auditability.

---

7) Calibration & feedback (metrics and how they adjust behavior)

Let ii index resolved questions, and yiy_i be the realized outcome.

7.1 Binary Brier score

Brierbinary = 1N∑i=1N(pi−yi)2,yi∈{0,1}.\text{Brier}_{\text{binary}} \;=\; \frac{1}{N}\sum_{i=1}^N \bigl(p_i - y_i\bigr)^2, \quad y_i\in\{0,1\}.

Lower is better. Decompose into reliability (calibration), resolution, and uncertainty if desired.

7.2 Multiclass Brier score (MCQ)

If pi=(pi1,…,piK)\mathbf{p}_i=(p_{i1},\dots,p_{iK}) and the realized class is k⋆k^\star, let yi\mathbf{y}_i be one-hot on k⋆k^\star. Then

BrierMC = 1N∑i=1N∑k=1K(pik−yik)2.\text{Brier}_{\text{MC}} \;=\; \frac{1}{N}\sum_{i=1}^N \sum_{k=1}^K \bigl(p_{ik}-y_{ik}\bigr)^2.

7.3 Expected Calibration Error (ECE)

Bin forecasts by their stated **confidence** (e.g., for MCQ, use top-1 probability). For bin bb with nbn_b items:

ECE = ∑bnbN | acc¯b−conf¯b |,\text{ECE} \;=\; \sum_{b} \frac{n_b}{N}\;\bigl|\;\overline{\text{acc}}_b - \overline{\text{conf}}_b \;\bigr|,

where acc¯b\overline{\text{acc}}_b is the empirical accuracy within the bin and conf¯b\overline{\text{conf}}_b the average predicted confidence. Lower is better (closer alignment).

7.4 CRPS for numeric

If FiF_i is the predictive CDF and the realized value is yiy_i,

CRPS(Fi,yi) = ∫−∞∞(Fi(x)−1{x≥yi})2 dx.\text{CRPS}(F_i, y_i) \;=\; \int_{-\infty}^{\infty}\bigl(F_i(x) - \mathbf{1}\{x\ge y_i\}\bigr)^2\,dx.

For mixture distributions, approximate by Monte Carlo from FiF_i. Lower is better.

7.5 Feedback loop

A separate calibration script computes these metrics **only for forecasts made before resolution** and writes a short **“Calibration Guidance”** memo (e.g., “slightly overconfident in 60–80% bin; reduce sharpness there”). Spagbot **injects this memo at the top of all future prompts**, nudging panel behavior; the math (Beta/Dirichlet/mixture) remains unchanged.

---

8) Practical notes on uncertainty, robustness, and edge cases

- **Weak priors** (α0,β0≪1\alpha_0,\beta_0\ll 1; α0k≪1\alpha_{0k}\ll 1) keep the combiner honest when only a few panelists respond.
- **Sampling**: for intervals and numeric percentiles, a few tens of thousands of draws are plenty; set a reproducible RNG seed.
- **Parsing failures / timeouts**: simply omit that model (i.e., wm=0w_m=0), which correctly widens posterior uncertainty.

- **Numeric tails**: the **mixture** preserves heavy tails if any forecaster projects them; avoid averaging percentiles directly (that can shrink tails artificially).
- **GTMC1 sensitivity**: since inputs are elicited, the Monte Carlo **jitter** is essential; the reported pgtmcp_{\text{gtmc}} is a **frequency** over perturbed worlds, not a single-run outcome.

One-page formula recap

- **Binary posterior:** Beta(α0+∑wmpm+wgtmcpgtmc, β0+∑wm(1−pm)+wgtmc(1−pgtmc))\text{Beta}\bigl(\alpha_0+\sum w_m p_m + w_{\text{gtmc}} p_{\text{gtmc}},\; \beta_0+\sum w_m(1-p_m)+ w_{\text{gtmc}}(1-p_{\text{gtmc}})\bigr).
- **MCQ posterior:** Dir(α0k+∑mwmpmk)k=1K\text{Dir}\bigl(\alpha_{0k}+\sum_m w_m p_{mk}\bigr)_{k=1}^K.
- **Numeric final:** mixture ∑mwmW N(μm,σm2)\sum_m \tfrac{w_m}{W}\,\mathcal{N}(\mu_m,\sigma_m^2) with (μm,σm)(\mu_m,\sigma_m) from quantile regression.
- **GTMC1 signal:** pgtmc≈1R∑r1{final median≥50}p_{\text{gtmc}} \approx \tfrac{1}{R}\sum_r \mathbf{1}\{\text{final median} \ge 50\}.
- **Calibration:** Brier (binary & multiclass), ECE (binned), CRPS (numeric).

That's the full mathematical picture of Spagbot: a **structured evidence → weak-prior Bayes → uncertainty-preserving** pipeline, with an optional strategic simulator that contributes a **Monte-Carlo-estimated** probability to the binary aggregator, and a **calibration memo** that tunes forecaster behavior over time without changing the core math.

File maps

Here's a clear, annotated "file map" for the core Spagbot pieces you asked about. For each file I show: what it's for, the major sections you'll see, a **search anchor** you can paste into VS Code's search box, and a rough "where in the file" pointer (top / middle / bottom or an approximate line block). Where it helps, I also note the most common edits you might make later.

---

spagbot.py (the conductor)

What it is: The end-to-end runner: pulls Metaculus questions, builds research briefs, queries the LLM ensemble, optionally runs GTMC1, aggregates with Bayes-MC, logs to CSV, and (optionally) submits forecasts.

High-level layout you'll see (top → bottom)

Header + imports + timezone helper

- **Anchor to search:** Spagbot — Ensemble (OpenRouter GPT-5 → fallback 4o + Gemini + Grok) + GTMC1 + Bayes-MC
- **Where:** very top (~lines 1–40).
- **Why it matters:** Quick summary of what this build does.

Configuration block (env toggles, keys, model IDs, timeouts, tournament, file paths)

- **Anchors:**
  - # Configuration
  - SUBMIT_PREDICTION / USE_OPENROUTER / USE_GOOGLE / ENABLE_GROK
  - OPENAI_API_KEY / GOOGLE_API_KEY / XAI_API_KEY
  - OPENROUTER_GPT5_THINK_ID / OPENROUTER_CLAUDE37_ID / GEMINI_MODEL
  - TOURNAMENT_ID / FORECASTS_CSV
- **Where:** top-quarter (~lines 60–210).
- **Typical edit:** switch models or toggle providers by changing the env vars these read.

Mini-bench bookkeeping (skip duplicates for "one-shot" mini tourneys)

- **Anchor:** ANCHOR: "MINI-BENCH one-shot helpers"
- **Where:** top-quarter (~lines 215–280).
- **Why:** Avoids multiple submissions to "mini" tournaments.

Calibration note loader (auto-injects into prompts)

- **Anchors:**
  - ANCHOR: "CALIBRATION: loader"
  - CALIBRATION_PATH = os.getenv("CALIBRATION_PATH", "data/calibration_advice.txt")
  - _CAL_PREFIX = ("CALIBRATION GUIDANCE…")
- **Where:** top-quarter (~lines 285–340).
- **Why:** Whatever update_calibration.py writes gets prefixed to all forecasting prompts here.

MCQ "wide" CSV helpers

- **Anchors:** MAX_MCQ_OPTIONS = 20, def _mcq_wide_headers(), write_mcq_wide_row(
- **Where:** upper-middle (~lines 345–420).
- **Why:** Keeps an analysis-friendly MCQ output table.

Metaculus API helpers and Market Snapshot (Metaculus + Manifold)

- **Anchors:**
  - def list_posts_from_tournament(/ get_open_question_ids_from_tournament(
  - def _metaculus_prob_yes_from_post(
  - def _manifold_top_match(
  - def build_market_consensus_snapshot(
- **Where:** middle (~lines 430–760).
- **Why:** pulls questions and builds that little "consensus box" appended to research.

Research pipeline (LLM researcher + AskNews caching)

- **Anchors:** async def run_research_async(, RESEARCHER_PROMPT, build_research_prompt(
- **Where:** middle (~lines 770–1010).
- **Typical edit:** you can tune word limits or headings inside RESEARCHER_PROMPT.

LLM ensemble setup and callers

- **Anchors:**
  - @dataclass class ModelSpec / DEFAULT_ENSEMBLE
  - _get_or_client() (OpenRouter), _call_openrouter(), _call_google(), _call_xai(
  - _call_one_model()
- **Where:** middle (~lines 1015–1275).
- **Typical edit:** add/remove models by editing DEFAULT_ENSEMBLE.

Parsers for outputs (binary %, MCQ vector, numeric percentiles)

- **Anchors:** _extract_last_percent, _parse_mcq_vector, _NUMERIC_PAT, _parse_numeric_pcts, _coerce_p10_p50_p90
- **Where:** middle (~lines 1278–1355).
- **Why:** Makes the strict end-lines machine-readable.

Forecasting prompts (Binary / Numeric / MCQ — with calibration prefix)

- **Anchors:** BINARY_PROMPT = _CAL_PREFIX + """, NUMERIC_PROMPT = _CAL_PREFIX + """, MCQ_PROMPT = _CAL_PREFIX + """
- **Where:** middle (~lines 1358–1625).
- **Typical edit:** wording of the instructions; keep the "Final:" lines/format intact.

GTMC1 helpers (is this strategic? actor extraction prompt + call)

- **Anchors:** STRATEGIC_KEYWORDS, looks_strategic(, build_actor_extraction_prompt(, extract_actors_via_llm(
- **Where:** middle (~lines 1628–1765).
- **Why:** Only fires for strategic binary questions.

Ensemble runners (one per question type)

- **Anchors:** run_ensemble_binary(/ run_ensemble_mcq(/ run_ensemble_numeric(
- **Where:** middle (~lines 1768–1845).
- **What they do:** call each model, parse, and pack member outputs.

CSV writers

- **Anchors:** AGG_HEADERS, ensure_csvs(, write_rows(, ensure_mcq_wide_csv(, write_mcq_wide_row(
- **Where:** middle (~lines 1848–1917).
- **Outputs:** forecasts.csv, forecasts_by_model.csv, forecasts_mcq_wide.csv.

Detailed per-run reasoning log builder

- **Anchor:** def append_detailed_forecast_log(
- **Where:** middle (~lines 1920–2015).
- **Output:** forecast_logs/<RUNID>_reasoning.log.

The core pipeline (per question)

- **Anchor:** async def forecast_one(
- **Where:** lower-middle (~lines 2018–2460).
- **Flow:** research → ensemble → (GTMC1 if strategic & binary) → Bayes-MC → CSV/logs → (optional) submit.
- **Helpful anchors inside:**

        - ANCHOR: "safe locals for numeric/discrete aggregation" (initializes variables safely)
        - GTMC1 activated: Yes (where the GTMC1 brief is injected and the second binary pass runs)
        - ANCHOR: "MCQ ordered vector payload"
        - ANCHOR: "numeric/discrete CDF builder (tournament-compliant)"
    - **Typical edit:** weighting, e.g., give extra weight to GTMC1 by adjusting the gtmc_base_w before it's passed to Bayes-MC.

Async runner + CLI

- **Anchors:** async def main_async(, def main():
- **Where:** bottom (~lines 2465–end).
- **What:** selects questions, creates the per-run log file, loops over forecast_one, handles --mode and --limit.

GTMC1.py (game-theoretic Monte Carlo)

What it is: A compact bargaining/coalition simulator inspired by BDM/Scholz. Takes an **actor table** (name, position, capability, salience, risk_threshold), runs many jittered simulations, and returns a probability-like **exceedance** signal plus diagnostics.

Key sections

Module docstring with outputs and usage

- **Anchor:** GTMC1 – Enhanced BDM/Scholz Monte Carlo with table input + logging
- **Where:** very top.
- **What to read:** explains exceedance_ge_50, runs_csv, etc.

Sampling helpers (adds realistic jitter per Monte Carlo run)

- **Anchors:** sample_position(, sample_capability(, sample_salience(, sample_risk(, sample_threshold(
- **Where:** top-quarter.
- **Typical edit:** widen/narrow variation (e.g., variation=5.0).

Actor dataclass

- **Anchor:** @dataclass class Actor: (with position, capability, salience, risk, challenge_threshold)

- **Where:** upper-middle.
- **Tip:** weight() is capability×salience; that's the influence proxy.

Utility functions for distances, utilities, and win probability

- **Anchors:** get_range(, get_median(, probability(, EU_challenge(, EU_challenge_reverse(
- **Where:** upper-middle.
- **What:** expected utility of challenging, with simple/bilateral framings.

Core dynamics — class BDMScholzEnhancedModel

- **Anchors:** update_dynamic_Q(, check_coalitions(, coalition_formation(, simulation_round(, run_model(
- **Where:** middle.
- **What:** iterates rounds; actors move toward weighted midpoints when mutual EU clears thresholds; gentle learning on risk and challenge_threshold; forms coalitions when positions cluster.

Table → Actors + Monte Carlo harness

- **Anchors:** _coerce_actor_row(, _actors_from_table(, run_monte_carlo_from_actor_table(
- **Where:** middle-to-lower.
- **What:** builds actors, runs num_runs, writes CSV (gtmc_logs/<timestamp>_<slug>_runs.csv) and a meta JSON; returns a compact signal dict.

CLI demo

- **Anchor:** def main(): (tiny hard-coded example)
- **Where:** bottom.
- **Use:** python GTMC1.py to sanity-check the simulator writes a CSV and prints a signal.

---

bayes_mc.py (Bayesian + Monte Carlo aggregator)

What it is: The statistical combiner for panel outputs: **Binary** (Beta), **MCQ** (Dirichlet), **Numeric** (mixture of normals fit to each model's P10/P50/P90). Returns samples and summary stats (mean, P10/50/90).

Key sections

Config + evidence containers

- **Anchors:**
  - DEFAULT_SAMPLES = 20000 / DEFAULT_SEED = 42
  - @dataclass class BinaryEvidence / MCQEvidence / NumericEvidence
- **Where:** top.
- **Why:** controls MC stability; NumericEvidence can take raw samples if you have them.

Binary updater (Beta)

- **Anchors:** @dataclass class BinaryPrior, def update_binary_with_mc(
- **Where:** upper-middle.
- **What:** fractional counts α+=w·p, β+=w·(1−p); returns posterior samples and p10/p50/p90.
- **Typical edit:** the prior — Spagbot passes a weak prior (α=β=0.1) from the runner.

MCQ updater (Dirichlet)

- **Anchors:** @dataclass class DirichletPrior, def update_mcq_with_mc(
- **Where:** middle.
- **What:** α_k += w·p_k, sample ϕ~Dirichlet(α); returns per-option means and intervals.

Numeric updater (mixture of normals)

- **Anchors:** _fit_normal_from_q(, def update_numeric_with_mc(
- **Where:** middle-to-lower.
- **What:** for each model, fit (μ,σ) from its P10/P50/P90; draw samples proportional to weight; concatenate mixture; summarize P10/P50/P90.
- **Why:** preserves inter-model disagreement vs. “averaging percentiles.”

Calibration update files (how “calibration guidance” flows)

There are **two places** to know:

1. **The script that builds the guidance** → update_calibration.py
2. **The text file Spagbot reads** → data/calibration_advice.txt (default path)

update_calibration.py (builds the memo)

What it is: Reads forecasts.csv, fetches resolution data from Metaculus, keeps **only forecasts made before resolution**, computes calibration for binary / MCQ / numeric, and writes a short human-readable memo to data/calibration_advice.txt. Safe to run even if nothing is resolved yet (it writes a neutral note).

Key sections to search:

Paths / outputs

- **Anchor:** CSV_PATH = "forecasts.csv" / OUT_DIR = "data" / OUT_PATH = os.path.join(OUT_DIR, "calibration_advice.txt")
- **Where:** top.
- **Edit:** change where the memo is written if needed.

Metaculus resolution accessors

- **Anchors:** _get_resolution_dt(, _resolve_binary_outcome(, _mcq_resolved_option(
- **Where:** upper-middle.
- **What:** robust ways to figure out resolution time and outcome from the question JSON.

Row parsers

- **Anchors:** _collect_mcq_probs(, _numeric_percentiles(
- **Where:** middle.
- **What:** read your CSV's MCQ columns and numeric percentiles flexibly.

Metrics

- **Anchors:** _ece_from_bins(, _brier_binary(, _brier_multiclass(, _crps_mc()
- **Where:** middle.
- **What:** ECE (top-1 MCQ), Brier (binary + multiclass), and a CRPS Monte Carlo for numeric.

Main builder and CLI

- **Anchors:** def build_calibration_note():, def main():
- **Where:** lower third.
- **What:** groups rows by URL, filters to pre-resolution, computes per-decile stats, writes the memo.

- **Run it:** python update_calibration.py (creates/overwrites data/calibration_advice.txt).

data/calibration_advice.txt (the memo Spagbot injects)

What it is: The short guidance text that gets **prefixed to every forecast prompt** (see CALIBRATION_PATH in spagbot.py). When you have enough resolved questions, re-run the script to refresh this file; Spagbot will pick it up next run.

- **Default contents (when nothing resolved yet):** brief "no data yet; keep conservative" tips.
- **Where it's loaded:** in spagbot.py under the **anchor** CALIBRATION: loader.

How they fit together (calibration loop)

1. You run Spagbot → it writes forecasts and metadata to forecasts.csv.
2. Later, when some questions have resolved, run python update_calibration.py → it recomputes calibration using only pre-resolution forecasts and writes data/calibration_advice.txt.
3. Next Spagbot run, the **Calibration loader** reads that memo (CALIBRATION_PATH) and **injects it** at the top of each prompt, nudging the panel to correct biases.

(Optional) Where dependencies are declared

If you need to know what packages these files rely on, check pyproject.toml (Python 3.11; key deps include forecasting-tools, openai, asknews, numpy, python-dotenv). Handy when setting up a fresh environment.

Quick "find it fast" cheat sheet (copy/paste into VS Code search)

- MINI-BENCH one-shot helpers → mini-bench guard (spagbot.py).
- CALIBRATION: loader → where the calibration note is loaded (spagbot.py).
- run_research_async(→ research brief pipeline (spagbot.py).
- build_market_consensus_snapshot(→ Metaculus/Manifold box (spagbot.py).
- run_ensemble_binary(/ run_ensemble_mcq(/ run_ensemble_numeric() → ensemble runners (spagbot.py).
- forecast_one(→ the whole per-question pipeline (spagbot.py).
- run_monte_carlo_from_actor_table(→ main GTMC1 entry (GTMC1.py).
- BDMScholzEnhancedModel → the bargaining stepper (GTMC1.py).

- update_binary_with_mc(/ update_mcq_with_mc(/ update_numeric_with_mc(→ Bayes-MC combiners (bayes_mc.py).
- build_calibration_note(→ computes the memo (update_calibration.py).
