Predicting User Decisions from Prediction Markets

Jack "Siggy" Sigler

April 21, 2026

Abstract

Personalization in large language models has primarily focused on text generation and preference alignment, rather than modeling how individuals make decisions. In this work, I study user-level downstream personalization as a decision prediction problem: given a user and a market context, can a model predict the user's action? I use prediction markets from Manifold as a testbed, where user behavior is sequential, timestamped, and accompanied by reasoning signals such as comments and stake sizes. I compare four personalization strategies: random sampling baseline (RSB), LLM prompting baseline (LPB), per-user LoRA adapters, and an agent-based approach. My results show that the agent-based approach performs best, achieving 67% accuracy compared to 48% for per-user adapters. In addition, when accounting for both training and inference, the agent-based method uses approximately 10× fewer tokens and is about 4× faster. These findings suggest that while baking user-specific data directly into model parameters is appealing, stronger results can be achieved using a context-driven agent. This approach not only improves accuracy, but is also more efficient in both speed and token usage.

1 Introduction

Effective collaboration between humans improves over time as individuals learn how each other think, weigh evidence, and make decisions. This shared understanding enables more efficient communication, reduces the need for repeated context, and allows collaborators to anticipate each other's actions. Current large language models lack this capability: they operate largely statelessly across interactions and do not build a persistent model of a specific user's decision process. As a result, human–model interaction remains fundamentally limited compared to human–human collaboration.

Large language models can solve a wide range of reasoning tasks, yet they do not adapt to the decision behavior of a specific individual after deployment [9]. Most personalization today remains limited to surface signals such as style choice or prompt retrieval, rather than learning how a person updates beliefs, weighs evidence, or converts uncertainty into concrete actions [16].

The personalization literature distinguishes between personalized **text generation** and **downstream personalization**, where the model improves performance on a task rather than the quality of text [16]. **User-level personalization** is the finest granularity in this space, where a model adapts to a single person rather than a population or persona [16]. Decision-making is a particularly compelling target for this level of personalization because it reflects reasoning under uncertainty rather than stylistic preference [2, 7].

Prediction markets such as Manifold provide a natural testbed for this goal. Market participation is inherently sequential, timestamped, and contextualized by reasoning signals such as comments, stake sizing, and timing. This structure enables evaluation of whether a model can internalize a user's decision policy and generalize it to new, unseen situations [3, 12].

2 Related Work

Existing research on prediction markets primarily studies market-level behavior, including efficiency, microstructure, and manipulation, rather than learning or predicting individual decision policies [4, 10, 11, 15]. Behavioral analyses in this domain are typically descriptive, focusing on aggregate patterns rather than modeling user-specific actions.

Personalization research in large language models has focused on text generation, preference modeling, and recommendation tasks [1, 14]. More recent approaches incorporate retrieval-based memory and parameter-efficient fine-tuning to adapt models to users [6, 13, 17]. However, these methods are typically evaluated on text similarity or preference alignment rather than the ability to predict user actions in a decision-making context. Approaches such as population-level fine-tuning and persona-level conditioning do not capture how a specific user's decision logic evolves over time [6, 13, 17].

Existing benchmarks for personalization measure consistency of generated text or alignment with stated preferences, but do not evaluate the accuracy of predicted actions in sequential decision settings [5, 14, 17]. As a result, there is no standard framework for assessing whether a model has learned an individual user's decision policy.

This setting can be viewed as learning a user-specific decision policy from behavioral data, which differs fundamentally from both preference modeling and language generation tasks.

3 Dataset

3.1 Source Data

All user decision data used in this project was sourced from the Manifold Markets public API [8]. Using the API, I collected user profiles, decision histories, and comments for all current users on the platform. In total, the dataset contains approximately 180,000 users, 9,500,000 user decisions (bets), and 560,000 comments spanning December 2021 to February 2026.

The data was normalized from the API into a local PostgreSQL database, which was used for storing and retrieving data throughout the project.

Users are represented by a unique identifier along with associated metadata such as username and bio. All decisions (bets) and comments are linked back to this user identifier. Prediction markets on Manifold are internally represented as *contracts*, which contain *bets, comments, descriptions, outcome_type (question type), and answer choices*. Manifold supports several market types, including *binary, multiple_choice, pseudo_numeric, number, stonk, poll, and bountied_question*; however, this project focuses only on *binary* and *multiple_choice* markets.

User decisions correspond to bets placed on markets, where users select an outcome and optionally stake currency to express confidence. In this work, I focus on predicting the direction of the bet

(i.e., the selected outcome) and its timing, rather than the magnitude of the wager.

Each training example is constructed as a tuple of the form *(contract_data, user_data, ground_truth_decision)*.

3.2 User Filtering and Cohort Construction

Since users with high bet and comment activity are the most informative for this task, I filter the dataset to a subset of *strongest_users*, which are the users used for long-horizon prediction experiments. These 15 users are selected by removing accounts tagged as bots by the platform and ranking the remaining users by total bet and comment activity.

Let $\mathcal{U}^\star$ denote this filtered set of strongest users. For each user $u \in \mathcal{U}^\star$, let their sequence of bets be ordered in canonical time:

$$(b_1^{(u)}, b_2^{(u)}, \ldots, b_{T_u}^{(u)}),$$

where T_u is the number of bets considered for user u (capped at 13,000).

Collectively, these strongest users account for 794,014 bets and 56,527 comments.

For each user u, this sequence is partitioned into three disjoint splits:

$$(b_1^{(u)}, \ldots, b_{T_u}^{(u)}) = \mathcal{D}_{\text{train}}^{(u)} \cup \mathcal{D}_{\text{val}}^{(u)} \cup \mathcal{D}_{\text{test}}^{(u)},$$

where the splits respect temporal ordering and follow a fixed ratio of 0.59/0.26/0.15.

These splits are constructed once per user and remain fixed across all methods to ensure consistent evaluation. Methods that do not require training simply ignore $\mathcal{D}_{\text{train}}^{(u)}$ and $\mathcal{D}_{\text{val}}^{(u)}$. All reported accuracy metrics are computed exclusively on $\mathcal{D}_{\text{test}}^{(u)}$.

4 Methods

4.1 Task Definition

As introduced in **Section 3.1**, the goal of a personalization method is to maximize its accuracy in reproducing the decisions of a specific user. Rather than predicting an objective outcome, the task is to match the user's actual behavior.

For a given user $u \in \mathcal{U}^\star$, the sequence of bets $(b_1^{(u)}, \ldots, b_{T_u}^{(u)})$ is considered in canonical time order, where each bet $b_i^{(u)}$ occurs strictly before $b_{i+1}^{(u)}$.

The evaluation proceeds sequentially through this timeline. At each step i, the method is asked to predict the decision associated with b_i, given only information that would have been available prior to that bet. This enforces a strict temporal constraint and prevents any form of future leakage.

More concretely, each prediction at step i is conditioned on two context sets:

$$\mathcal{U}_i \quad \text{(user context)} \qquad \text{and} \qquad \mathcal{C}_i \quad \text{(contract context)}.$$

- $\mathcal{U}_i$ contains information derived from the user's past behavior (e.g., prior bets, comments, or aggregate statistics).
- $\mathcal{C}_i$ contains information about the current contract associated with b_i (e.g., market metadata, question text, or state).

Each prompting-based method receives the same static $\mathcal{C}_i$, which includes:

- **question:** the market question text
- **description:** additional context provided by the market
- **outcome_type:** the type of decision space (e.g., binary, multiple-choice)
- **mechanism:** the market mechanism used
- **volume:** total trading volume
- **unique_bettor_count:** number of distinct participants

The exact construction of $\mathcal{U}_i$ depends on the specific method (e.g., retrieval-based, parameter-adapted, or agent-based). However, two constraints are enforced across all methods:

1. **Temporal gating:** Both $\mathcal{U}_i$ and $\mathcal{C}_i$ are restricted to information available strictly before time step i. No future data is ever accessible.
2. **Contract isolation:** Let c_i denote the contract associated with bet b_i. Then $\mathcal{U}_i$ excludes all prior interactions between the same user and c_i. That is, even if the user has made earlier bets on c_i, these are not included in $\mathcal{U}_i$. This prevents trivial signal leakage in cases where users split a single decision into multiple smaller, repeated bets.

At each step, the method produces a prediction $\hat{y}_i$ for the user's true decision y_i, conditioned on $(\mathcal{U}_i, \mathcal{C}_i)$. Performance is measured by accuracy over the full sequence:

$$\frac{1}{T}\sum_{i=1}^{T}\mathbf{1}[\hat{y}_i = y_i]$$

This framing treats personalization as a sequential prediction problem over user decisions, where success is defined by the ability to consistently replicate an individual user's choices over time.

4.2 Random Sampling Baseline (RSB)

The RSB is a simple statistical baseline that assigns a uniform probability over all possible decisions for each bet $b_i^{(u)}$. Let $|\mathcal{Y}_i|$ denote the number of possible decisions for the contract associated with $b_i^{(u)}$. Then the reported RSB accuracy for each prediction is $\frac{1}{|\mathcal{Y}_i|}$. The RSB requires no training and does not utilize any data from $\mathcal{D}_{\text{train}}^{(u)}$ or $\mathcal{D}_{\text{val}}^{(u)}$.

4.3 LLM Prompting Baseline (LPB)

The LPB is the first method that makes use of both $\mathcal{U}_i$ and $\mathcal{C}_i$ to produce user decision predictions. For each $b_i^{(u)}$, the LPB dynamically inserts this context into a master prompt (full prompt provided in **Appendix A.1.1**) and asks the model to predict the decision made by the user. I used **Gemini 3.1 Pro Preview** for the LPB. A full list of the API hyperparams used in the experiment can be found in **Appendix A.1.2**.

The LPB is restricted to "static" context information. Specifically, it receives the common $\mathcal{C}_i$ defined in **Section 4.1**, along with the following user-specific attributes for $\mathcal{U}_i$:

- **username:** the user's display name
- **bio:** the user's profile description
- **balance:** current account balance
- **total_profit:** cumulative profit across all bets
- **total_volume:** total amount wagered
- **follower_count:** number of followers

The model is then asked to output a predicted decision outcome, which is extracted using a simple best-effort parser.

The LPB receives a score of 1 for $b_i^{(u)}$ if the parsed predicted outcome matches the ground truth decision, and 0 otherwise. A score of 0 is also assigned if the parsed prediction is not an element of $\mathcal{Y}_i$.

4.4 Per-User LoRA (PUL)

The PUL method builds directly upon the LPB by retaining a similar inference-time prompting mechanism while introducing a per-user adapter-based training procedure. This method uses **Qwen2.5-7B-Instruct** as the base model with LoRA adapters implemented via PEFT. Full training hyperparameters are provided in **Appendix A.2.2**.

For training, each user $u \in \mathcal{U}^\star$ is assigned a persistent set of LoRA weights that capture their individual decision patterns. The model is trained on the user's $\mathcal{D}_{\text{train}}^{(u)}$ split. For each $b_i^{(u)}$, a prompt is constructed similarly to the LPB, but the user context $\mathcal{U}_i$ is extended using retrieval over the user's history.

In particular, $\mathcal{U}_i$ is augmented with:

- **Relevant user comments:** past comments with high semantic similarity to the current contract context
- **Relevant user bets:** past decisions with high semantic similarity to the current contract context

The model is trained to predict the user's decision given $(\mathcal{U}_i, \mathcal{C}_i)$, with loss computed on the predicted outcome tokens. Adapters are trained for 10 epochs, and checkpoint selection is performed using performance on $\mathcal{D}_{\text{val}}^{(u)}$.

Additional details on the retrieval-augmented prompt are provided in **Appendix A.2.1**, and a full description of the training and retrieval system is given in **Section 5**.

4.5 Agent-Based Method (ABM)

While the PUL approach extends the LPB through a training procedure, the ABM instead focuses on enhancing the inference-time prediction process. In particular, it expands the effective context for $\mathcal{U}_i$ and introduces a time-gated context compaction and retrieval mechanism. The ABM uses **Gemini 3.1 Pro Preview** as the base model. Full prompts and hyperparameters are provided in **Appendix A.2.1**.

As with previous methods, predictions are conditioned on the context tuple $(\mathcal{U}_i, \mathcal{C}_i)$. In the ABM, this context is constructed dynamically through an exploration phase, during which the model can issue tool calls to gather additional user-specific information.

The available tools include:

- **Recent user bets:** the user's most recent prior bets (one per contract) before the prediction time
- **User bet statistics:** aggregate summaries of the user's historical behavior, including counts and answer distributions
- **Relevant user bets:** past bets by the same user with high semantic similarity to the current contract
- **Relevant contract comments:** prior comments on the current contract with high semantic similarity to the agent's query

Through this exploration process, the model incrementally builds an enriched $\mathcal{U}_i$. Once the exploration phase is complete, the agent produces a final prediction using the same output format as the other methods.

4.6 Methods Summary

Table 1 consolidates the four method families before the discussion turns to the infrastructure that supports them in **Section 5**.

Method	Model	User Context	Training	Personalization Mechanism
RSB	None	None	No	Uniform random sampling over valid outcomes.
LPB	Gemini 3.1 Pro Preview	Static profile fields and aggregate user metadata	No	One-shot prompting over shared contract context and static user attributes.
PUL	Qwen2.5-7B-Instruct + PEFT LoRA	Static user data plus retrieved bets and comments	Yes	Train a persistent per-user adapter on retrieval-augmented prompts.
ABM	Gemini 3.1 Pro Preview	Tool-built user context plus retrieval and compacted memory	No	Dynamically gather and compress user evidence at inference time.

Table 1: Methods summary for the four personalization approaches

5 System Architecture

5.1 System Summary

The overall system is organized around a CPU-side orchestration layer that owns data access, retrieval, experiment control, and logging, while specialized external services handle model execution and agent-side tool use. PostgreSQL serves as the shared store for normalized market data, user histories, and embedding-backed retrieval state, allowing every method to operate over the same leakage-safe source of truth.

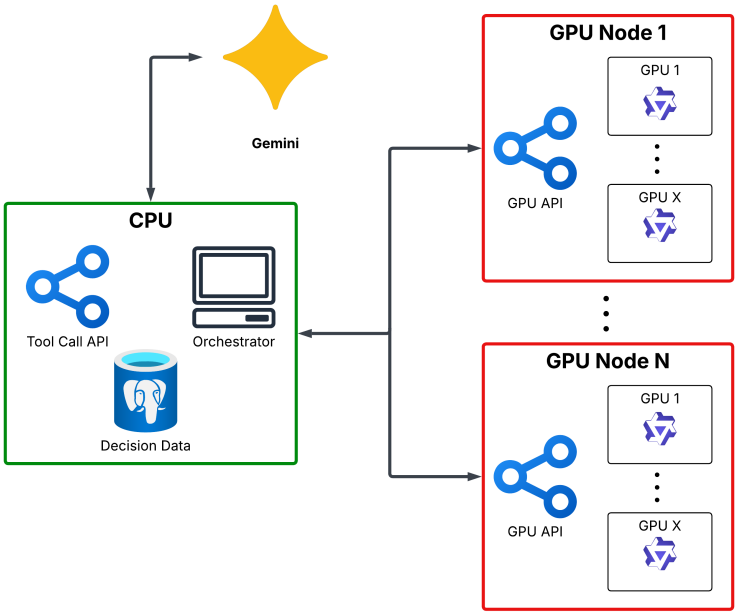

Figure 1: High-level summary of the experimental system architecture

Figure 1 summarizes this design. The CPU-side orchestrator constructs time-gated inputs from the database, dispatches model work to remote GPU nodes through a lightweight GPU API, and exposes a constrained tool-call API for the agent-based method. This separation keeps the data and evaluation logic centralized on the CPU side while allowing training and inference to scale across multiple GPU workers.

5.2 RAG Retrieval

To enrich the user context $\mathcal{U}_i$, I introduced a semantic retrieval system over each user's historical bets and comments. For users in $\mathcal{U}^\star$, contract-question embeddings for past bets and text embeddings for past comments are precomputed with **BAAI/bge-m3** and stored in PostgreSQL using **pgvector**. At inference time, the system reuses the most recent stored embedding for the target contract and retrieves the top-k most similar historical bets and top-k most similar historical comments for user u, subject to strict time gating ($created\_time < before\_time$) and exclusion of the current contract to prevent leakage.

To keep the retrieved history within the prompt budget, I apply a greedy context-packing scheme. Each retrieved item is assigned an approximate token cost based on its serialized JSON length, and the total context budget is split evenly between bet history and comment history. The highest-similarity items that fit are selected from each group first; any remaining budget is then filled using the highest-similarity leftover items from either group. Long comments may also be truncated before packing.

5.3 Distributed Training Network

To train the persistent per-user adapters used by the PUL method, I built a distributed training network that separates CPU-side orchestration from GPU-side model execution. The experiment

orchestrator and data layer run on a CPU machine connected to PostgreSQL, while a pool of remote GPU workers hosts the base **Qwen2.5-7B-Instruct** model together with its LoRA adapter runtime. Because the base model fits on a single A100 GPU, parallelism is achieved across users rather than within a single model replica: each worker trains one user-specific adapter at a time, allowing multiple (u, worker) pairs to progress in parallel.

To reduce network overhead, the base model and active adapter state remain resident on the GPU worker during training. For each epoch, the CPU orchestrator assigns a user $u \in \mathcal{U}^\star$ to an available worker, constructs the time-gated retrieval-augmented prompts on the CPU, and sends the resulting supervised training examples to the worker through a lightweight GPU API. The worker then performs the local LoRA update steps, while the CPU side retains responsibility for scheduling, data access, split management, and artifact logging.

The same worker-level pinning pattern is reused at inference time in the persistent evaluation pipeline. Once a user $u \in \mathcal{U}^\star$ is assigned to a worker, that worker processes the full sequential evaluation for that user, keeping the relevant adapter checkpoint resident on the paired GPU worker while the CPU side constructs prompts, issues API calls, and logs artifacts. This avoids repeatedly shuffling adapter state across workers during evaluation and keeps the runtime behavior aligned with the training architecture.

With this execution model in place, training proceeds on a per-user basis with validation driven updates after each epoch. Where the same worker evaluates the adapter on that user's validation split using deterministic decoding. Concretely, the worker sweeps a fixed scale grid $\lambda \in \{0.0, 0.33, 0.66, 1.0\}$, applies each scale to the adapter during batched validation inference, and selects the best-performing value $\lambda_t^\star$, with ties broken in favor of the larger scale. If this validation score improves on the user's previous best, that epoch's adapter state is promoted to the user's durable `best` checkpoint, which is the only best-checkpoint state later used for persistent evaluation. The orchestrator also records the associated best epoch, selected $\lambda_t^\star$, and full per-λ validation metrics in the user artifacts so that the chosen checkpoint can be reproduced and reloaded exactly. This scale controls how strongly the learned LoRA adapter influences generation relative to the base model, so the sweep serves as a lightweight calibration step after every epoch rather than treating adapter strength as a fixed constant.

Rather than baking $\lambda_t^\star$ directly into the checkpoint, the system smooths it across epochs with an exponential moving average,

$$\bar{\lambda}_t = 0.3\lambda_t^\star + 0.7\bar{\lambda}_{t-1},$$

with a minimum value of 0.1. This prevents large epoch-to-epoch swings in adapter strength and yields more stable checkpoints. Early stopping is applied at the user level. A user is retired if validation fails to produce a new best checkpoint for three consecutive epochs. Training can also stop once the smoothed scale has effectively converged, defined as a spread of less than 0.05 across the three most recent EMA values. Together, these rules limit unnecessary training once a user-specific adapter has stabilized, while still allowing the system to checkpoint and resume each user's state without retransmitting full model weights.

To make this concrete, I now show how the smoothed scale $\bar{\lambda}_t$ is incorporated directly into the LoRA parameters and how it affects the model weights. Let A_t and B_t denote the LoRA matrices after epoch t. I bake the EMA scale into the adapter by defining:

$$A_t^\star = \sqrt{\bar{\lambda}_t}\, A_t, \qquad B_t^\star = \sqrt{\bar{\lambda}_t}\, B_t$$

so that the effective low-rank update becomes

$$B_t^\star A_t^\star = \bar{\lambda}_t \, B_t A_t.$$

The model weights at time t are then:

$$W_t = W_{\text{base}} + \delta \, B_t^\star A_t^\star$$

where $A_t^\star, B_t^\star$ are the scaled matrices and

$$\delta = \frac{\alpha}{r}$$

is the standard LoRA scaling factor, with α the LoRA scaling hyperparameter and r the rank.

- $\bar{\lambda}_t$ controls *time-varying strength* of the adapter
- δ sets the *overall scale* of LoRA updates

This shows that $\bar{\lambda}_t$ directly modulates the magnitude of the low-rank update applied to the base model at each epoch.

5.4 Agent Infrastructure

In order to support the ABM, I designed a lightweight tool-call API together with a context compactor. The tool API is implemented as a simple microservice that gives the agent guardrailed access to the PostgreSQL-backed user history and embedding indexes. Since the database is quite large, and similarity search can quickly incur large costs, the tool layer enforces strict request bounds on listing and retrieval operations. More importantly, it also enforces the same temporal gating used throughout the evaluation: the agent may only access information available before the current prediction step i.

Architecturally, the tool API expands $\mathcal{U}_i$ without changing $\mathcal{C}_i$. The contract context $\mathcal{C}_i$ remains the same static contract payload used by the other prompting methods, while the agent uses tool calls to gather additional user-specific evidence for $\mathcal{U}_i$, such as recent bets, aggregate betting statistics, semantically similar prior bets, and semantically similar prior comments on the target contract. This keeps the ABM comparable to the other methods while allowing $\mathcal{U}_i$ to be constructed dynamically at inference time.

In an effort to still allow the agent to have long-range memory of a user across thousands of historical actions, while preserving proper time gating and contract isolation, I introduced a context compactor. The compactor maintains a leakage-safe approximation to a much larger user history and injects that approximation back into future prompts as part of $\mathcal{U}_i$.

The compactor operates as a three-layer growing window of context:

1. **Per-Contract Event Ledgers:** Raw, time-gated bet and comment events from the user history prior to $b_i^{(u)}$, stored separately by contract. These ledgers are part of the internal state used to build $\mathcal{U}_i$, but they are never injected directly into the prompt.
2. **Future-Safe Checkout Buffers:** Events promoted out of the ledgers once a contract is "closed" for that user, meaning there is no later bet by the same user on that contract. This promotion rule enforces contract isolation and makes the buffered events safe to reuse in future $\mathcal{U}_j$ contexts.

3. **Compacted Prompt Memory:** Two bounded text summaries, one for bets and one for comments, produced from the future-safe buffers by the LLM compactor. This is the only compactor layer injected into later agent prompts, and therefore the only compactor-managed component that directly augments $\mathcal{U}_i$.

Operationally, every k rounds the compactor checks which Layer 1 contracts have no future user bets remaining and moves those events into Layer 2. Once Layer 2 accrues enough future-safe context, the system issues a compaction request to the same base agent model, updating Layer 3 according to

$$\texttt{compact}(\texttt{old\_memory} + \texttt{future\_safe\_events}).$$

In this way, the ABM can carry forward a compact approximation to long-range user behavior without directly exposing raw historical sequences in the prompt window. The specific compaction prompts are provided in **Appendix A.3.2**, along with the full compactor hyperparameters used in the experiment.

As in the persistent adapter architecture, the inference runtime is also organized around stable per-user CPU-side execution. Once a user is assigned to an evaluation thread, that thread owns the agent loop, tool client, and compactor state for the full sequential rollout over that user's bets, while the tool API remains an external read-only service boundary. This avoids rebuilding compactor state across rounds and preserves a consistent notion of $\mathcal{U}_i$ as the user timeline advances.

6 Experimental Setup

All methods were evaluated on the fixed strongest-user cohort $\mathcal{U}^\star$ described in **Section 3.1**, consisting of 16 users, 794,014 bets, and 56,527 comments after filtering. For each user $u \in \mathcal{U}^\star$, the chronologically ordered bet sequence was split once into $\mathcal{D}^{(u)}_{\text{train}}$, $\mathcal{D}^{(u)}_{\text{val}}$, and $\mathcal{D}^{(u)}_{\text{test}}$ using a fixed ratio of 0.59/0.26/0.15, and these splits were held constant across all methods. All reported results are computed on $\mathcal{D}^{(u)}_{\text{test}}$ under the shared sequential evaluation protocol from **Section 4.1**: at step i, each method predicts the decision for $b^{(u)}_i$ using only information available before that bet, with strict temporal gating and contract isolation. The primary metric is exact-match accuracy, with runtime and token-usage comparisons reported where relevant.

7 Results

7.1 Training

Persistent training ran for 253.8 hours, or roughly 10.6 days, across the strongest-user cohort. For the training diagnostics in this subsection, I exclude one corrupted user artifact, leaving 15 analyzable users. The resulting training trajectories show substantial heterogeneity in how long different users remain worth training. As shown in **Figure 2**, 10 of 15 users reached their best validation score by epoch 3, and 14 of 15 did so by epoch 4. However, the median user still spent roughly 43% of total training time after its best epoch, suggesting that the current early-stopping policy is conservative and allows substantial post-peak training.

The validation-time lambda scan also shows meaningful user-level variation. The best validation lambda values were distributed across $\lambda \in \{0.33, 0.66, 1.00\}$ with counts $\{4, 6, 5\}$ respectively, and

no user preferred $\lambda = 0$. This indicates that every analyzable user benefited from some non-zero amount of adapter scaling, but that the amount of useful adaptation was not constant across the cohort.

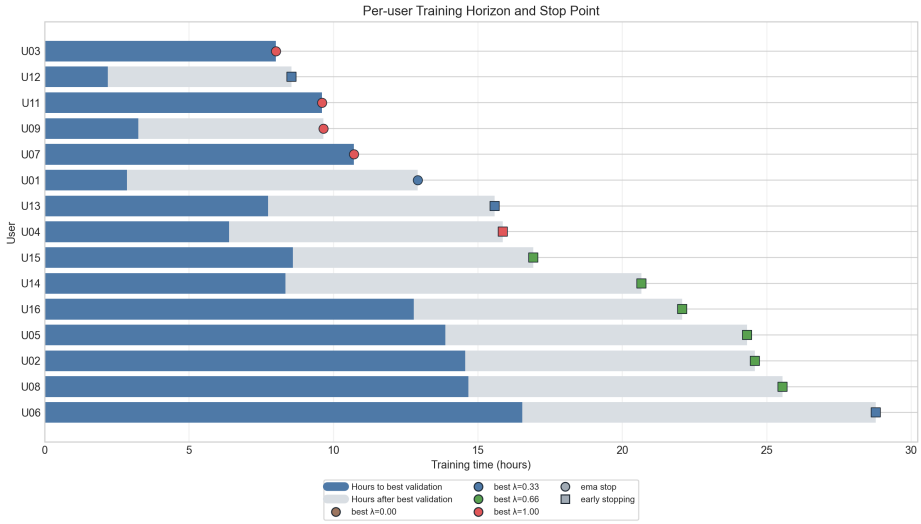

Figure 2: Per-user training time to peak validation performance

Figure 3 compares each user's best validation score with that same user's PUL experiment score. The relationship is strong. The Pearson correlation is $r = 0.836$. On average, experiment scores were lower than best validation scores by 0.053 absolute accuracy, so validation is not perfectly calibrated. However, the user ordering transfers well enough that validation performance is clearly informative about experiment-time performance. This is the main evidence that the validation-time lambda scan was selecting real signal rather than fitting noise.

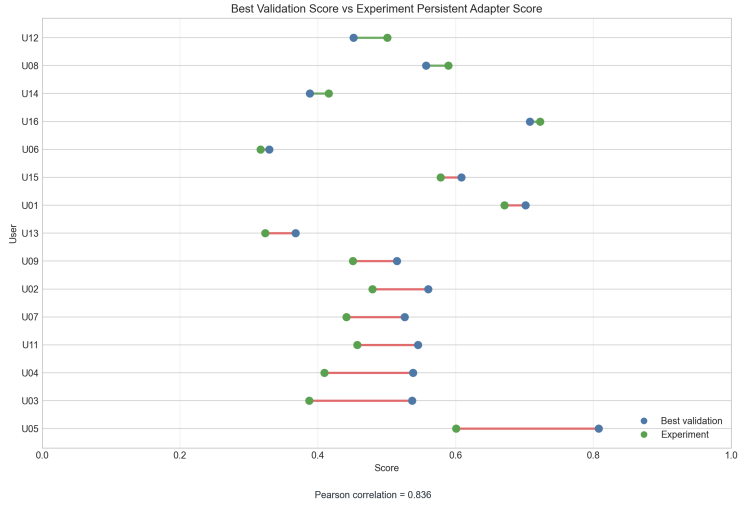

Figure 3: Best validation score versus PUL experiment score for each user.

7.2 Primary Comparison

The long-horizon experiment evaluated each method on 27,910 user-rounds over 167.6 hours. Across the run, 465 rounds failed, yielding a successful-round rate of 98.3%. The dominant failure modes were GPU memory errors for long adapter contexts and transient Gemini API failures. Despite those failures, the run was stable enough to support a direct comparison across methods.

The headline accuracy comparison is shown in **Figure 4**. The Agent-Based Method (ABM) is the clear winner at 67.3% accuracy, compared with 48.9% for PUL, 48.4% for the LLM Prompting Baseline (LPB), and 40.0% for the Random Sampling Baseline (RSB). This is an 18.4 percentage point gain over PUL. The more interesting secondary result is that PUL essentially matches the LPB overall, outperforming it by 0.55 percentage points despite using Qwen 7B as its base model. On the 15-user subset retained for user-level diagnostics, the ABM is the best method for 14 users, while PUL outperforms the LPB for 10 users.

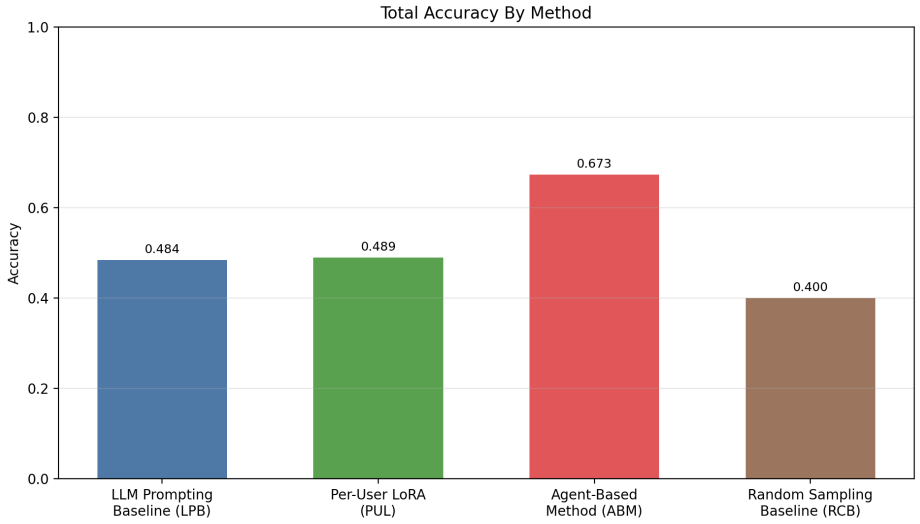

Figure 4: Overall accuracy by method on $\mathcal{U}^\star$.

7.3 Resource Tradeoffs

The full timing and token breakdowns are provided in **Appendix B**. Those appendix figures show that the ABM also dominates PUL on measured experiment-time cost once training is included. PUL requires 2.15× more wall-clock time and 4.49× more tokens than the ABM, with the vast majority of that cost coming from training rather than inference. Under this accounting, PUL is both less accurate and more expensive than the ABM in the long-horizon setting.

That said, the resource accounting in those figures is incomplete in an important way. PUL is built on a self-hosted open model, so after training it does not incur per-query API charges in the way that the LPB or ABM do. By contrast, the LPB and ABM are externally billed at inference time, and any deployment decision would need to account for that recurring API cost even though it is not visible in the token and wall-clock plots alone. I also do not observe the upstream training cost of Gemini itself. That cost is surely enormous, but it is outside the scope of what can be measured from the experiment logs. The fairest interpretation is that the appendix figures compare visible experiment-time usage rather than full lifecycle cost.

If I isolate inference-time usage, the comparison changes direction. PUL uses only 16,211 seconds and 45.2 million tokens during evaluation, while the ABM uses 433,015 seconds and 220.4 million tokens. This makes the ABM roughly 26.7× slower and 4.87× more token-intensive at prediction time. In other words, the ABM wins under full experiment accounting because PUL pays a very large upfront training cost. Once that training cost is sunk, PUL is the much cheaper runtime method.

7.4 Per User Difficulty

Per-user performance is not uniform across the strongest-user cohort. Averaging across the three non-random methods produces a clear shared difficulty ordering. Per-user average accuracy ranges from 0.406 to 0.717 with a standard deviation of 0.113. The easiest user for the model-based methods is BTE at 0.717, while the hardest is Ziddletwix at 0.406. Figure 5 shows that this is not just noise in one method. Users that are easy for one model tend to be easy for the others as well.

That shared structure is reflected in the cross-method correlations. Across users, the LPB and PUL have correlation $r = 0.788$, the LPB and ABM have $r = 0.721$, and PUL and the ABM have $r = 0.744$. Even the RSB is positively correlated with the learned methods, which suggests that some users are intrinsically easier because their forecasting histories induce more predictable evaluation rounds. The main conclusion is that user identity creates a real difficulty axis that is largely shared across methods, even though the ABM still dominates on absolute accuracy.

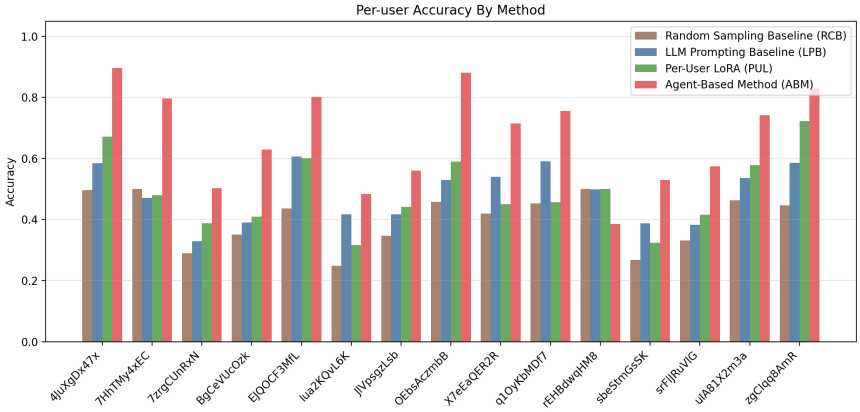

Figure 5: Per-user accuracy by method on the strongest-user cohort.

To understand what drives that shared difficulty, I joined the experiment scores with static user features and ran a small post-hoc strongest-user analysis. Simple behavior statistics already carry substantial signal. The strongest positive correlation with average non-random accuracy is the number of binary bets ($r = 0.793$), followed by comment count ($r = 0.598$) and binary-bet share ($r = 0.571$). The strongest negative correlation is multiple-choice bet share ($r = -0.608$). Intuitively, users who mostly trade binary markets appear easier to personalize than users whose activity is concentrated in multiple-choice markets.

A four-feature ridge regression using these static features explains a substantial share of the across-user variance. For the non-random average score target, the model reaches $R^2 = 0.773$ with adjusted $R^2 = 0.682$ and RMSE $= 0.052$. The learned coefficients in **Table 2** are consistent with the correlation analysis. More binary betting and more commenting push predicted performance up,

while heavier multiple-choice usage pushes it down. I do not treat this as a causal model, but it is strong evidence that a large fraction of per-user difficulty is predictable from coarse properties of a user's market history.

Feature	**Coefficient**
Binary bets	0.054
Comments	0.041
Total bets	-0.006
Multiple-choice bet share	-0.026

Table 2: Standardized ridge-regression coefficients for predicting per-user accuracy from static behavioral features.

8 Limitations

The main limitation of this study is that I was not able to run two clean control baselines because my research group ran out of our GCP credits. First, I do not have a context-dump control for the ABM. This leaves an important ambiguity in the strongest result of the paper. The LPB already tells me what a frontier model can do without tool use, but it does not answer whether the ABM wins because it reasons about what context to retrieve or simply because tool use plus a large context window lets it see much more user history. A useful follow-up would be a non-agent prompting method that receives a very large prompt built from recent and retrieved evidence without iterative tool use. If that method matched the ABM on Gemini, the conclusion would shift away from agentic retrieval itself and toward the claim that, for modern frontier models, giving the model much more context is already enough. Running the same context-dump control on Qwen would then answer a different question, namely whether this strategy also helps a smaller open model. If both Gemini and Qwen improved substantially, that would suggest that simply providing more context is broadly useful. If the gain appeared only for Gemini, that would suggest that smaller open models are more vulnerable to context pollution and still benefit from tighter retrieval control, whereas newer frontier models may need that control less.

Second, I do not report a clean non-adapter Qwen baseline in the experiment results, and this missing control affects how the adapter results should be interpreted. If a plain Qwen prompting baseline were already close to the LPB, then the PUL training procedure would provide limited benefit despite its substantial upfront cost. I do not believe this is the most likely explanation. In prior work, I ran a separate experiment where the adapter was trained and evaluated on the fly without using fixed splits. This experiment also included a larger set of users, each with fewer bets per user than in the main experiment presented in this paper. It ran over 24,950 bets and used the same inference-time prediction mechanism. In that setting, the Qwen baseline achieved 44.1% accuracy, the Gemini baseline achieved 51.5%, and the LoRA method achieved 53.0%, once again surpassing the Gemini baseline.

These results do not come from the same evaluation setting, so they should not be interpreted as a direct estimate of the missing persistent-experiment control. However, they make it plausible that a similar performance gap between Qwen and frontier models also existed in the main experiment. Under this interpretation, the fact that PUL roughly matches the LPB is meaningful, as it suggests that fine-tuning can close a substantial baseline gap between a smaller open model and a frontier

prompting baseline. The absence of this control therefore does not invalidate the adapter results, but it does prevent a precise measurement of how much of the observed gain is attributable specifically to adaptation.

The final major limitation of this work lies in the dataset itself. Although I made a strong effort to restrict the analysis to users who appear to be genuine rather than automated accounts, it is still possible that some portion of the decision data was generated by bots. This is partly acceptable for the purposes of this study. The task is to model and predict user behavior, and even if some of that behavior originates from automated agents, the methods are still learning consistent decision patterns. To the extent that bot behavior reflects structured strategies or heuristics, it can still contribute meaningful signal for prediction.

However, this remains a fundamental limitation. The ultimate goal of this work is to model human decision-making, and the presence of bot-generated data introduces a mismatch between the target objective and the underlying dataset. While the results suggest that the methods capture real behavioral structure, it is unclear how fully these findings transfer to purely human settings.

9 Conclusion

The central result of this paper is that inference time reasoning with access to user specific context was more effective than trying to compress user behavior into a persistent adapter. The ABM can decide what information to retrieve, how to combine recent and semantically similar evidence, and when to rely on broader contract context rather than a fixed behavioral summary. That flexibility appears to matter for this task. User decisions in prediction markets are conditional on evolving market state, recent evidence, and local framing, so a method that can actively inspect context at prediction time has a natural advantage over one that must express the user through a static parameter update.

At the same time, the adapter results should not be read as a failure of personalization. The adapter roughly matches the LPB overall, despite using a much smaller open model, and the validation-to-experiment correlation indicates that the training-time model-selection procedure was tracking real signal rather than noise. The non-zero spread in preferred λ values across users also suggests that there is no single universal adaptation strength. Some users benefit from stronger parameter updates, while others are better served by a more conservative mixture with the base model. In that sense, the training results support the broader claim that user-specific decision structure exists and can be learned. They mainly show that a simple PUL is not the strongest way to exploit that structure in this setting. The training diagnostics also suggest that this upfront adapter cost can be reduced. Most users reached their best validation score within four epochs and still spent a median of roughly 43% of total training time after that peak, so more aggressive early stopping could likely cut training cost without much loss in quality.

The per-user difficulty analysis reveals that variation in user performance is largely driven by task composition rather than purely intrinsic behavior. While users who are easy for one method tend to be easy for others, much of this variation is explained by coarse features, particularly the distribution of market types. Users with a higher proportion of binary bets are consistently easier to predict than those engaging more with multiple-choice markets. This suggests that observed differences in personalization performance are significantly influenced by the underlying prediction task, and future work should explicitly account for this when evaluating user-level difficulty.

Taken together, these findings point toward a hybrid view of personalization. Parameter adaptation remains useful because it can encode durable user tendencies in a reusable form. However, this benchmark suggests that personalization of decision-making also requires dynamic access to current context and the ability to retrieve the right evidence at the right time. For sequential forecasting behavior, the strongest systems may therefore be those that combine lightweight persistent user modeling with flexible inference-time retrieval and reasoning rather than relying on either approach alone.

References

[1] Personachat / convai2 datasets. Competition dataset page, 2018. URL https://convai.io/data/.

[2] Suhana Bedi, Hejie Cui, Miguel Fuentes, Alyssa Unell, Michael Wornow, Juan M. Banda, Nikesh Kotecha, Timothy Keyes, Yifan Mai, Mert Oez, Hao Qiu, Shrey Jain, Leonardo Schettini, Mehr Kashyap, Jason Alan Fries, Akshay Swaminathan, Philip Chung, Fateme Nateghi, Asad Aali, Ashwin Nayak, Shivam Vedak, Sneha S. Jain, Birju Patel, Oluseyi Fayanju, Shreya Shah, Ethan Goh, Dong-han Yao, Brian Soetikno, Eduardo Reis, Sergios Gatidis, Vasu Divi, Robson Capasso, Rachna Saralkar, Chia-Chun Chiang, Jenelle Jindal, Tho Pham, Faraz Ghoddusi, Steven Lin, Albert S. Chiou, Christy Hong, Mohana Roy, Michael F. Gensheimer, Hinesh Patel, Kevin Schulman, Dev Dash, Danton Char, Lance Downing, François Grolleau, Kameron Black, Bethel Mieso, Aydin Zahedivash, Wen-wai Yim, Harshita Sharma, Tony Lee, Hannah Kirsch, Jennifer Lee, Nerissa Ambers, Carlene Lugtu, Aditya Sharma, Bilal Mawji, Alex Alekseyev, Vicky Zhou, Vikas Kakkar, Jarrod Helzer, Anurang Revri, Yair Bannett, Roxana Daneshjou, Jonathan Chen, Emily Alsentzer, Keith Morse, Nirmal Ravi, Nima Aghaeepour, Vanessa Kennedy, Akshay Chaudhari, Thomas Wang, Sanmi Koyejo, Matthew P. Lungren, Eric Horvitz, Percy Liang, Mike Pfeffer, and Nigam H. Shah. Medhelm: Holistic evaluation of large language models for medical tasks. *arXiv preprint*, arXiv:2505.23802, 2025. URL https://arxiv.org/abs/2505.23802. Version v2, revised 2 Jun 2025.

[3] David D. Bourgin, Joshua C. Peterson, Daniel Reichman, Stuart J. Russell, and Thomas L. Griffiths. Cognitive model priors for predicting human decisions. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97, pages 5133–5141, 2019. URL https://arxiv.org/pdf/1905.09397.

[4] Constantin Bürgi, Wanying Deng, and Karl Whelan. Makers and takers: The economics of the kalshi prediction market. *CESifo Working Paper Series / Discussion Paper DP20631*, (12122), 2025. URL https://ssrn.com/abstract=5502658.

[5] Xiangnan et al. Geng. P5: Personalized preference pathways for recommendation. In *KDD 2023*, 2023.

[6] Wenkai Li, Jiarui Liu, Andy Liu, Xuhui Zhou, Mona Diab, and Maarten Sap. Big5-chat: Shaping llm personalities through training on human-grounded data. *arXiv preprint arXiv:2410.16491*, 2024. URL https://arxiv.org/abs/2410.16491.

[7] Jay Mandi, Bryan Wilder, Taylor Killian, et al. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *arXiv preprint arXiv:2307.13565*, 2023.

[8] Manifold Markets. Manifold markets api documentation. https://docs.manifold.markets/api, 2026. Accessed: 2026-04-08.

[9] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, and .... Gpt-4 technical report. *arXiv preprint*, arXiv:2303.08774, 2023. URL https://arxiv.org/abs/2303.08774. Version v6, last revised 4 Mar 2024.

[10] Nahid Rahman, Joseph Al-Chami, and Jeremy Clark. Sok: Market microstructure for decentralized prediction markets (depms). *arXiv preprint arXiv:2510.15612*, 2025. URL https://arxiv.org/abs/2510.15612.

[11] Itzhak Rasooly and Roberto Rozzi. How manipulable are prediction markets? *arXiv preprint arXiv:2503.03312v1*, 2025. URL https://arxiv.org/html/2503.03312v1.

[12] David M. Rothschild and Rajiv Sethi. Trading strategies and market microstructure: Evidence from a prediction market. *The Journal of Prediction Markets*, 10(1):1–29, 2016. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2322420. SSRN working paper version available.

[13] Zhaoxuan Tan, Qingkai Zeng, Yijun Tian, Zheyuan Liu, Bing Yin, and Meng Jiang. Democratizing large language models via personalized parameter-efficient fine-tuning. *arXiv preprint arXiv:2402.04401*, 2024. URL https://arxiv.org/abs/2402.04401.

[14] Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. Personalizing dialogue agents: I have a dog, do you have pets too? In *ACL*, 2018. URL https://arxiv.org/abs/1801.07243.

[15] Xu Zhang, Yiming Zhang, and Xiang et al. Li. Agent-based modelling for real-world stock markets under behavioral economic principles. *arXiv preprint arXiv:2307.12987*, 2023. URL https://arxiv.org/abs/2307.12987.

[16] Zhehao Zhang, Ryan A. Rossi, Branislav Kveton, Yijia Shao, Diyi Yang, Hamed Zamani, Franck Dernoncourt, Joe Barrow, Tong Yu, Sungchul Kim, Ruiyi Zhang, Jiuxiang Gu, Tyler Derr, Hongjie Chen, Junda Wu, Xiang Chen, Zichao Wang, Subrata Mitra, Nedim Lipka, Nesreen Ahmed, and Yu Wang. Personalization of large language models: A survey. *arXiv preprint*, arXiv:2411.00027v3, 2025. URL https://arxiv.org/pdf/2411.00027.

[17] Zheng Zhao, Clara Vania, Subhradeep Kayal, Naila Khan, Shay B. Cohen, and Emine Yilmaz. Personalens: A benchmark for personalization evaluation in conversational ai assistants. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 18023–18055, 2025. URL https://aclanthology.org/2025.findings-acl.927.

A Method Details

A.1 LLM Prompting Baseline (LPB)

A.1.1 LPB Prompt

The LLM Prompting Baseline (LPB) uses the following baseline prompt template, with the JSON placeholders populated at runtime from the user profile, contract metadata, and allowed answer

choices.

```
You are modeling a specific user on a prediction market. Your task is to predict the single outcome this

You must output exactly one JSON object predicting which answer choice this user would select.

Rules:
1. Use ONLY the provided user context and contract context.
2. You MUST select exactly one answer from the allowed choices.
3. Your entire output MUST be a single JSON object with ONE key.
4. Do NOT add text, explanations, Markdown, or extra keys.
5. The value MUST match exactly one of the allowed answer choices (uppercase).

Required output format (match EXACTLY):
{"predicted_outcome": "<ONE ALLOWED CHOICE>"}

User Context:
{user_profile_json}

Contract Context:
{contract_json}

Question:
{contract_question}

Allowed Answer Choices (uppercase):
{answer_choices_json}

Reminder - final output MUST be exactly:
{"predicted_outcome": "<ONE ALLOWED CHOICE>"}
```

A.1.2 LPB Hyperparameters

The LPB uses a Gemini-backed client with the following active settings:

- **model:** `gemini-3-pro-preview`
- **base_url:** `https://generativelanguage.googleapis.com/v1beta`
- **temperature:** `0.7`
- **max_generation_attempts:** `3`
- **max_output_tokens:** `None`
- **request_timeout_s:** `60`
- **est_chars_per_token:** `4`

A.2 Per-User Adapter

A.2.1 Adapter Prompt

The adapter uses the following retrieval-augmented prompt template, with the JSON placeholders populated at runtime from the user profile, retrieved history, contract metadata, and allowed answer choices.

```
You are modeling a specific user on a prediction market. Your task is to predict the single outcome this

You must output exactly one JSON object predicting which answer choice this user would select.

Rules:
1. Use ONLY the provided user context, history context, and contract context.
2. You MUST select exactly one answer from the allowed choices.
3. Your entire output MUST be a single JSON object with ONE key.
4. Do NOT add text, explanations, Markdown, or extra keys.
5. The value MUST match exactly one of the allowed answer choices (uppercase).

Required output format (match EXACTLY):
{"predicted_outcome": "<ONE ALLOWED CHOICE>"}

User Context:
{user_profile_json}

Relevant User History (bets and comments):
{history_context_json}

Contract Context:
{contract_json}

Question:
{contract_question}

Allowed Answer Choices (uppercase):
{answer_choices_json}

Reminder - final output MUST be exactly:
{"predicted_outcome": "<ONE ALLOWED CHOICE>"}
```

A.2.2 Adapter Hyperparameters

The persistent-experiment adapter was evaluated with the following worker configuration:

- **base model:** `Qwen/Qwen2.5-7B-Instruct`
- **adapter rank r:** `16`
- **adapter scaling α:** `16`
- **LoRA dropout:** `0.2`
- **target modules:** `q_proj, v_proj`

- **training dtype:** `bfloat16`
- **learning rate:** `5e-5`
- **training prompt mode:** retrieval-augmented (`rag`)
- **retrieval top-k:** `5`
- **total context token budget:** `3400`
- **maximum comment characters per retrieved event:** `900`
- **train / validation / test split ratios:** `0.59 / 0.26 / 0.15`
- **lambda during training:** `1.0`
- **lambda during evaluation:** `1.0`
- **sampling enabled:** `true`
- **generation temperature:** `0.7`
- **generation top-p:** `0.9`
- **maximum new tokens:** `256`

A.3 Agent-Based Method (ABM)

A.3.1 ABM Prompt

The ABM uses a two-stage prompt. During exploration, the model may call a constrained set of tools to gather more context. When the tool or step budget is exhausted, the runner switches to a final-answer prompt that only allows the `submit_prediction` tool. The placeholders are populated at runtime from the user profile, contract metadata, allowed answer choices, compacted bet/comment memory, and the tool results accumulated so far.

Exploration prompt

```
You are an agent that predicts which outcome a specific user would choose on this contract.

You may request tool calls to gather more context. Use ONLY the tools listed below.
Time gating is handled internally; do not provide absolute timestamps.
Recency should be controlled only by limit/top_k, not by time-window parameters.
When you are ready to answer, call ‘submit_prediction‘ with the predicted outcome.

Available Tools:
- list_user_bets_recent(limit<=15)  # current user only
- get_user_bet_stats()  # current user only
- search_user_bet_embeddings(query_text, top_k<=15, filters? contract_id only)  # current user only
- search_contract_comment_embeddings(contract_id, query_text, top_k<=15)
- submit_prediction(predicted_outcome)  # Use only when ready to answer.

Tool call format (exact JSON):
{"tool_call": {"name": "<TOOL_NAME>", "arguments": "<JSON STRING>"}}

User Context:
```

```
{user_profile_json}

Contract Context:
{contract_json}

Question:
{contract_question}

Allowed Answer Choices (uppercase):
{answer_choices_json}

Compacted User Memory (Bets):
{compacted_bets_memory}

Compacted User Memory (Comments):
{compacted_comments_memory}

Context Gathered So Far:
<<CONTEXT_SO_FAR>>
```

Final-answer prompt

```
You are an agent that predicts which outcome a specific user would choose on this contract.

You must answer by calling ‘submit_prediction‘. No other tool calls are allowed.

User Context:
{user_profile_json}

Contract Context:
{contract_json}

Question:
{contract_question}

Allowed Answer Choices (uppercase):
{answer_choices_json}

Compacted User Memory (Bets):
{compacted_bets_memory}

Compacted User Memory (Comments):
{compacted_comments_memory}

Context Gathered So Far:
<<CONTEXT_SO_FAR>>

Return ONLY this JSON (exact format):
{"tool_call": {"name": "submit_prediction", "arguments": "{\"predicted_outcome\": \"<ONE ALLOWED CHOICE
```

A.3.2 ABM Compactor Prompts

The ABM uses two internal compaction prompts, one for bets and one for comments. These prompts are not shown to the model during tool use; they are used only by the compactor when it compresses future-safe event buffers into bounded bet-memory and comment-memory text blocks.

Bet compaction prompt

```
You are maintaining a compact behavioral memory of a user's betting history on a prediction market plat
This memory will be injected into future prompts to help predict how the user will bet on new contracts

Your task: merge the existing memory with new bet events into a single updated memory block.

Focus on preserving:
- Recurring topics, markets, or contract types the user bets on
- Directional tendencies (does the user tend to bet YES/NO, favor underdogs, follow consensus?)
- Bet sizing patterns if meaningful (large vs small amounts)
- Any notable individual bets worth remembering (unusual size, strong conviction)

Rules:
1. Use only facts from the input - never invent details.
2. Be concise. Prefer patterns over exhaustive lists.
3. Output must not exceed {max_chars} characters.
4. Return plain text only - no JSON, no markdown, no headers.
5. If current_memory is empty, summarize only the new events.

Current memory:
{current_memory}

New bet events:
{events}
```

Comment compaction prompt

```
You are maintaining a compact behavioral memory of a user's comment history on a prediction market plat
This memory will be injected into future prompts to help predict how the user will bet on new contracts

Your task: merge the existing memory with new comment events into a single updated memory block.

Focus on preserving:
- Topics and domains the user engages with most
- The user's reasoning style (data-driven, intuitive, contrarian, deferential to experts?)
- Confidence and tone patterns (hedged vs assertive, how often they update views)
- Any specific strong opinions or domain expertise that appears repeatedly

Rules:
1. Use only facts from the input - never invent details.
2. Be concise. Prefer patterns and characterizations over quoting comments verbatim.
3. Output must not exceed {max_chars} characters.
4. Return plain text only - no JSON, no markdown, no headers.
5. If current_memory is empty, summarize only the new events.

Current memory:
```

```
{current_memory}

New comment events:
{events}
```

A.3.3 ABM Configuration

The persistent-experiment ABM was evaluated with the following active agent and compactor settings:

- **agent model:** `gemini-3-pro-preview`
- **base_url:** `https://generativelanguage.googleapis.com/v1beta`
- **temperature:** `0.7`
- **max_output_tokens:** `7000`
- **request_timeout_s:** `60`
- **max_steps:** `4`
- **max_tool_calls:** `2`
- **max_context_chars:** `8000`
- **tool limits:** recent-bet listing `limit <= 15`; embedding search `top_k <= 15`
- **compactor checkout cadence:** every `5` steps over batches of `256` contracts
- **comment fetch limit per refresh:** `500`
- **compaction thresholds:** bets after `7` events; comments after `5` events or `1500` chars
- **per-event truncation:** bets/question text `400` chars; comments `600` chars
- **compacted memory budgets:** bets `4000` chars; comments `6000` chars

B Additional Resource Tradeoffs

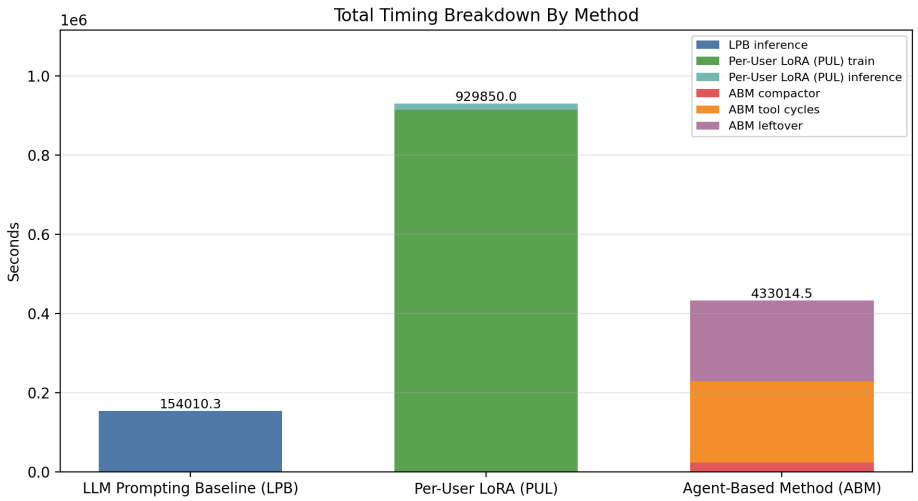

Figure 6: Measured total wall-clock time by method, including compactor time, tool-cycle time, and remaining loop overhead for the ABM stack.

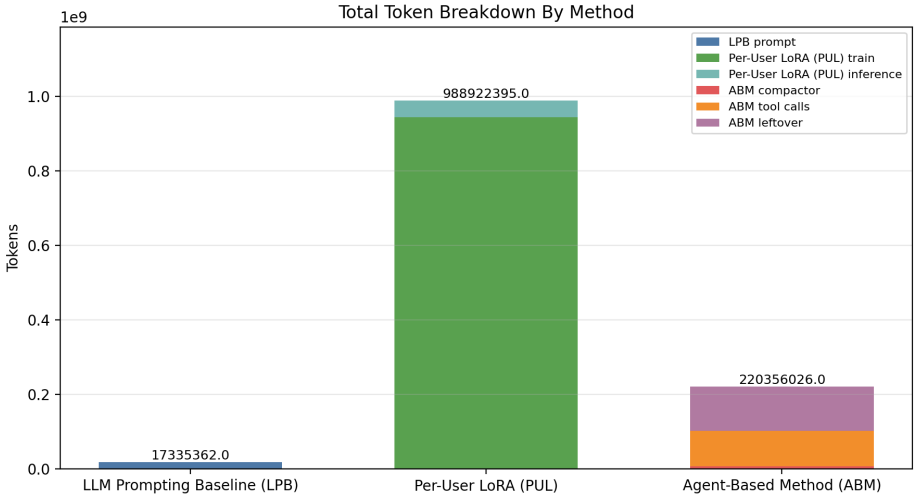

Figure 7: Measured total token usage by method, including compactor tokens, tool-call tokens, and remaining model tokens for the ABM stack.