Tiktok Post Analytics Predictor

1 User and Decision

Target user: TikTok content creators or brands wanting to maximize engagement (likes, comments, shares) on their posts.
Decision: Whether to post the content as it is, or make small adjustments such as to the timing of the post, photo content, or caption tone before publishing to improve engagement.

2 Target and Horizon

Target: A numeric engagement level such likes, comments, follows, or shares on a post.
Timeline: Engagement within the first 24 hours of posting.
Type: Done by regression on account-specific weights to determine engagement score.

3 Features (No Leakage)

Post metadata: Caption sentiment analysis and length, posting time of day, hashtags Photo analysis of content: Content, People, Photo Quality. Account history, analysis of engagement of past posts, posting frequency, comment analysis and what people want to see.
Exclusions: Prediction time leakage: Future engagement data such as likes, comments, or information attained after posting. For example, after the user presses a post, the API should not use its current engagement stats to factor in improvements to the post. Training Time Leakage: During training, we cannot use any data that occurred after a training post was published in order to predict it because then it may suffer leakage. When deciding a training set, we must make sure future posts never leak backwards in training.

4 Baseline $\rightarrow$ Model Plan

Baseline: A rule-based model that returns the moving average of engagement (likes) for the user's last 5 posts can be a suitable minimal benchmark that would be simple to implement and require little amounts of data. The assumption here is that the user's current post will be similar to their recent posts in engagement. This can be helpful as future models should exceed the performance of this baseline as they will have more access to data.

5 Metrics, SLA, and Cost

Metrics: Since we are using regression, Mean Absolute Percent Error (MAPE) could be an appropriate way of measuring the degree of accuracy of the model in a human readable way. Since it reports things as percentages, it would be useful to see how well the model performs across different magnitudes (small content creators vs. larger content creators). However, for predictions with small engagement, predicting 10 likes vs. 1 like, then you can have crazy huge errors (900%). To counteract this, we will also track Mean Absolute Error (MAE) as a secondary metric, which is more stable for small number cases. SLA: p95 latency $<$ 500ms per prediction request. Cost envelope = free-tier hitting 100 requests/day and should automatically scale at around $<$ \$200/day at 50k requests/hour spikes. (More benchmarking and research is required to get a more accurate representation of costs and max costs)

6 Minimal Evaluation Plan

Train our model while avoiding leakages (done by splitting recent posts as our test set, while keeping historical posts as our training set).
Use Mape as our primary metric to optimize for during training, and MAE as a potentially secondary metric. Then compare the trained model against our moving average baseline to see which performs better.
Measure the end-to-end request time from a client request to a response. Can be done by looking at the Networks tab on the browser developer console for some sample real world use case timings.
Test normal use cases like 100 req/day which should sufficiently meet our p95 $<$ 500ms. Do some spike testing for 50k req/hour for a short amount of time to see how the system reacts. Measuring the time taken for each request to calculate other p50/p95/p99.
For costs, estimate reads/writes per request against the Database with user information. As well as invocation counts + execution time + memory allocation in GB-seconds for each request which can be used to calculate actual costs for running each lambda request.

7 API Design + Architecture

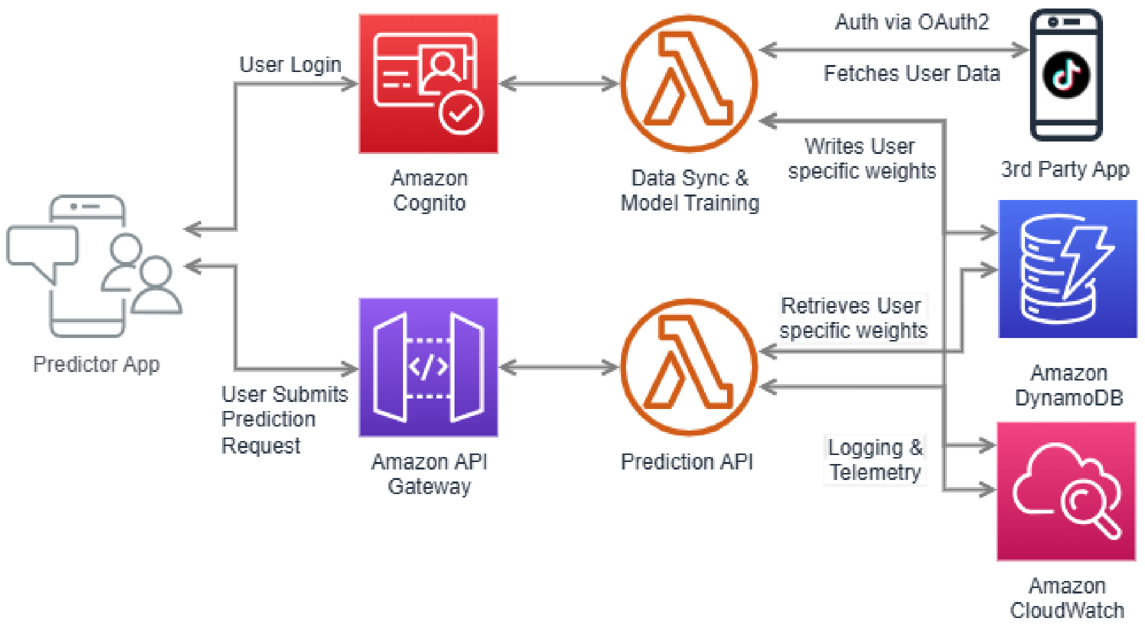

Figure 1: API Stack For Inference

- API Endpoint: **/predict** endpoint sits on AWS API Gateway and handles POST requests to initiate inference against our models via AWS Lambdas. Post requests are more secure than GET requests and also allows us to better structurally embed metadata like text and images represented in binary using the **multipart/form-data** data type.

  Sample Request Payload:

```
{
  "account_id": "acct_12345",
  "post": {
    "text": "Sunset at the pier #sunset #photography",
    "scheduled_time_utc": "2025-09-20T18:00:00Z",
    "hashtags": ["sunset","photography"],
    "location": {"lat":49.2827,"lon":-123.1207},
    "metadata": {"client":"web-v0.9","platform":"ios"}
  },
  "explain": true
}
```

Sample 200 OK response:

```
{
  "model_version":"v1.3.0",
  "prediction": {
    "predicted_engagement": 123.4,
    "predicted_bucket":"medium"
  },
  "suggestions":[
   {"feature":"post_time","action":"move_to","value":"2025-09-20T19:00:00Z","impact_estimate_pct":12},
    {"feature":"caption_sentiment","action":"increase_positive","impact_estimate_pct":6}
  ],
  "diagnostics":{
    "latency_ms": 312,
    "feature_fetch_ms": 45,
    "inference_ms": 180,
  }
}
```

- Login and Authentication: Authentication will be performed with **Amazon Cognito** that will use the **OAuth2** protocol to link with the user's account and fetch data. Cognito stores identities and credentials and will provision the keys to access a user's specific model weights that have been derived from their account post history. On first login and each subsequent login, a background update pipeline is triggered. Cognito → Lambda initiates a data fetch from the 3rd-party API (e.g., TikTok) to collect recent post history. This data is either processed directly in the Lambda or enqueued to an asynchronous worker (via SQS or Step Functions) to update per-user calibration weights. Updated weights are stored in DynamoDB and are used by the prediction Lambda on subsequent **/predict** calls. This separation keeps the actual prediction path lightweight (meeting p95 latency) while allowing our personalization to evolve with new data.

- Predictor API Endpoint Infrastructure: The two key infrastructures to consider are AWS Lambdas vs Fargate. For a prototype startup that handles only 100 requests a day, it makes sense to use Lambdas. As serverless environments that only run on demand when a user makes a request, this makes them very cost effective because we only pay for useful work. Since Fargate is a containerized environment, it incurs costs continuously to keep it alive. For a small startup, we will be paying for a lot of idle time which is a lot of wastage and will eat into our Free quota.

  However, the major argument for using Fargate is the high performance latency of a cold start for lambdas. ML models are often quite large (hundreds to thousands of megabytes), which can take quite a bit of time to load from a cold start and could potentially be a major bottleneck against our p95 latency target. Since Fargate is constantly kept warm, we would be able to cut out the cold-start latency. In addition, while baking models into lambda environments works for smaller less complex models, if we plan on supporting larger dynamic model loading from S3, each lambda instance may incur a cost where loading the models and keeping them in memory for containers might be more cost effective. This makes Fargate an enticing option to switch to once we build up a sufficient customer base to amortise the constant overhead and to scale to larger and more complex functionality in each individual request.

In the event of a request spike when our product goes viral hitting 50,000 req/hour, the elasticity of lambdas are considerably greater than Fargate containers. Lambdas automatically scale with requests and AWS handles load balancing, making them very flexible. The only major concern being configurable concurrency limits which can be adjusted. Fargate Containers on the other hand require smart autoscaling rules and intelligent logging to detect surges and launch extra containers as a response. However, these containers often have significant startup times and can take up to minutes to be ready. This delay can cause request throttling and possibly dropped requests if the response is not agile enough. In addition, overreacting with containers can potentially become expensive as many containers sit idle, while underreacting can cause requests to be dropped. Seeing as this is the first time we are handling a viral spike, it may be unrealistic to have the perfect scaling protocol amides uncertain data readings. Thus as an overall pick, it makes sense to stick with AWS lambdas as our starting architecture, but strongly consider migrating to Fargate containers if users stick around after the viral spike as returning customers.

- Telemetry: Data and telemetry is important to monitor logs, metrics, and alarms. Cloudwatch is a native AWS tool that integrates into Lambdas and will allow us to monitor latency times to hit our p95 metrics, as well as track costs and errors. Ensuring that the appropriate adjustments are made to ensure product stability, and keep under budget.

8 PIA excerpt: Privacy, Ethics, Reciprocity

8.1 Data Inventory

- User identity tokens: Cognito + OAuth2, for auth only.
- Extracted post metadata: captions, timestamps, likes/comments counts, attached media.
- User-specific model weights and preference settings.
- Model telemetry: request latency, error rates, invocation counts, billing metrics.
- Excluded Data: Direct Messages, Non-public posts (friend only), emails or phone numbers.

8.2 Purpose Limitation

- Data collected is used only for model training, personalization, and system monitoring.
- No secondary use such reselling to advertisers without explicit new consent from users.

8.3 Retention and Access

- Retain telemetry logs for 90 days for dev ops and debugging.
- Retain per-user calibration weights only while account is active. This means all data related to a user is deleted on account deletion.
- Ensure only authorized members of the team have access to customer data. Leave a trail about who and when data is accessed.

8.4 Telemetry decision matrix

Data type	Value	Invasiveness	Effort	Decision
User specific model weights	Very High	Medium	Requires DB storage	Keep
p95 latency logs	High	Low	Automatic via CloudWatch	Keep
User engagement predictions vs actuals	High	Medium	Requires opt-in feedback	Keep
Full raw posts (videos, captions)	Medium	High	Requires storage and privacy risk	Discard after feature extraction

8.5 Guardrails

- Opt-ins and disclosure: user must consent to share their social media history via terms of service. Illustrate clear disclosure on what is collected, how long it is retained, and how it is used.
- K-anonymity: Aggregate data collected for training purposes can be required to have at least k entries before being considered. Bucketing and adding noise to collected data can help prevent users from being reverse engineered and identified.

8.6 Reciprocity

- Value to users: users get personalized predictions, suggestions to improve engagement, and feedback dashboards based on their specific past data.
- Value to the platform: Users create more engaging posts that drive traffic and possibly increase profit from advertisements.

9 Risks and Mitigations

- Technical: Lambdas have cold starts and use global models that require a new build to update. Cold starts can increase prediction latency and outdated models can be inaccurate. This could lead to bad user experiences which can lead to user churn. Mitigation includes scheduling periodic retraining of global model with updated data, and customer feedback loop for reporting bad suggestions. For cold starts, we should test with synthetic loads to confirm SLA (p95 ¡ 500 ms) and consider switching to containerized solutions once the business scales.
- Legal: The livelihood of our business is coupled to the TOS of 3rd-party sites. Pulling user and engagement data from TikTok APIs may violate terms of service and be seen as unfair if used to 'game the algorithm' and commercialize their data. At any moment, the site can change their TOS and obstruct our operations. Mitigation includes keeping the 3rd-party legal teams in the loop with our activities, and use only official APIs with proper OAuth2 tokens. From a long-term strategy perspective, a formal partnerships to integrate the tool into app may be mutually beneficial.
- Ethical: Incentivizing homogenized content where users may rely too heavily on predictions, chasing trends rather than innovating. This could reduce creative diversity and reinforce popularity bias. Mitigation includes communicating that suggestions are advisory, not prescriptive. Potentially experimenting with "exploration mode" that suggest off trend creative ideas.