Currently I am working as a Machine Learning Engineer at Talentium. Talentium is an early-stage pre-seed startup. It's headquartered in Switzerland,and we raised our pre-seed funding round in July. Our goal is to revolutionize the recruitment process, by enabling companies and recruiters to find the best candidates, and filter through their profiles in an extremely short period of time. I was the first engineering hire there - well, not exactly the first, a web developer was hired before me, but I've been playing a fundamental role in setting up and scaling the ML infrastructure, developing the ML models and doing some research as well. I implemented an LLM-based real-time search engine, reducing search times and enabling our clients to find the right talent, faster. Further, I have developed a Conversational-AI system for HR inquiries [regarding candidates], and it has tangibly increased our user engagement by 33% and skyrocketed satisfaction levels for our AI-powered responses, seeing a surge of 40%.

Before working here, I worked as a Machine Learning Engineer in the R&D division of ScienceLogic for (about) 20 months. ScienceLogic is a high-growth, series E (late stage) startup, it plays a significant role in the realm of MLops and IT infrastructure monitoring (tools) [basically optimizing IT operations through cutting-edge technology and automation]

During my tenure there, I spearheaded the enhancement of our anomaly detection system which deals with an immense scale — we're talking about monitoring up to 10,000 devices every minute using the SL1 (ScienceLogic monitoring) software. One of my other notable contributions was the revamp of our hyperparameter optimization process [shift from grid-search to Optuna, developing a LightGBM module], which led to a significant [61%] reduction in our ML regression testing time and improved the speed of our algorithms as well.

I also interacted with our customers regularly - during our yearly symposium, and also while helping the platform engineering team troubleshoot customer issues. These interactions really enriched my perspective and helped me to develop better features based on the feedback.

Additionally, I had the opportunity to mentor a summer intern. Together, we leveraged MLflow to improve our ML performance and regression testing. This streamlined our processes so effectively that following a pull request on Bitbucket, once we had a successful build on Jenkins, it would automatically trigger the ML regression tests, and keep track of the results from different algorithms [everything ran like clockwork].

Working at a pre-seed startup has been an invaluable experience, allowing me to wear multiple hats, drive innovation, and contribute directly to the product's growth and development. The dynamism and agility of the startup environment have sharpened my problem-solving skills and deepened my hands-on expertise in machine learning.

However, I am at a stage in my career where I am seeking to align my passion and skills with an organization that has a significant impact on a global scale. I believe I am ready for a Senior role as I have consistently demonstrated the ability to take ownership of projects, collaborate with cross-functional teams, interface with customers, and mentor junior team Members.

QUESTIONS:

How does Samsara differentiate itself from other IoT companies in terms of its product and company culture?"

"What is the vision for the AI team in the next 3-5 years, and how does it align with the overall company objectives?"

"Given the recent awards, what practices or initiatives does Samsara believe contributed most to its recognition in culture, values, and growth?"

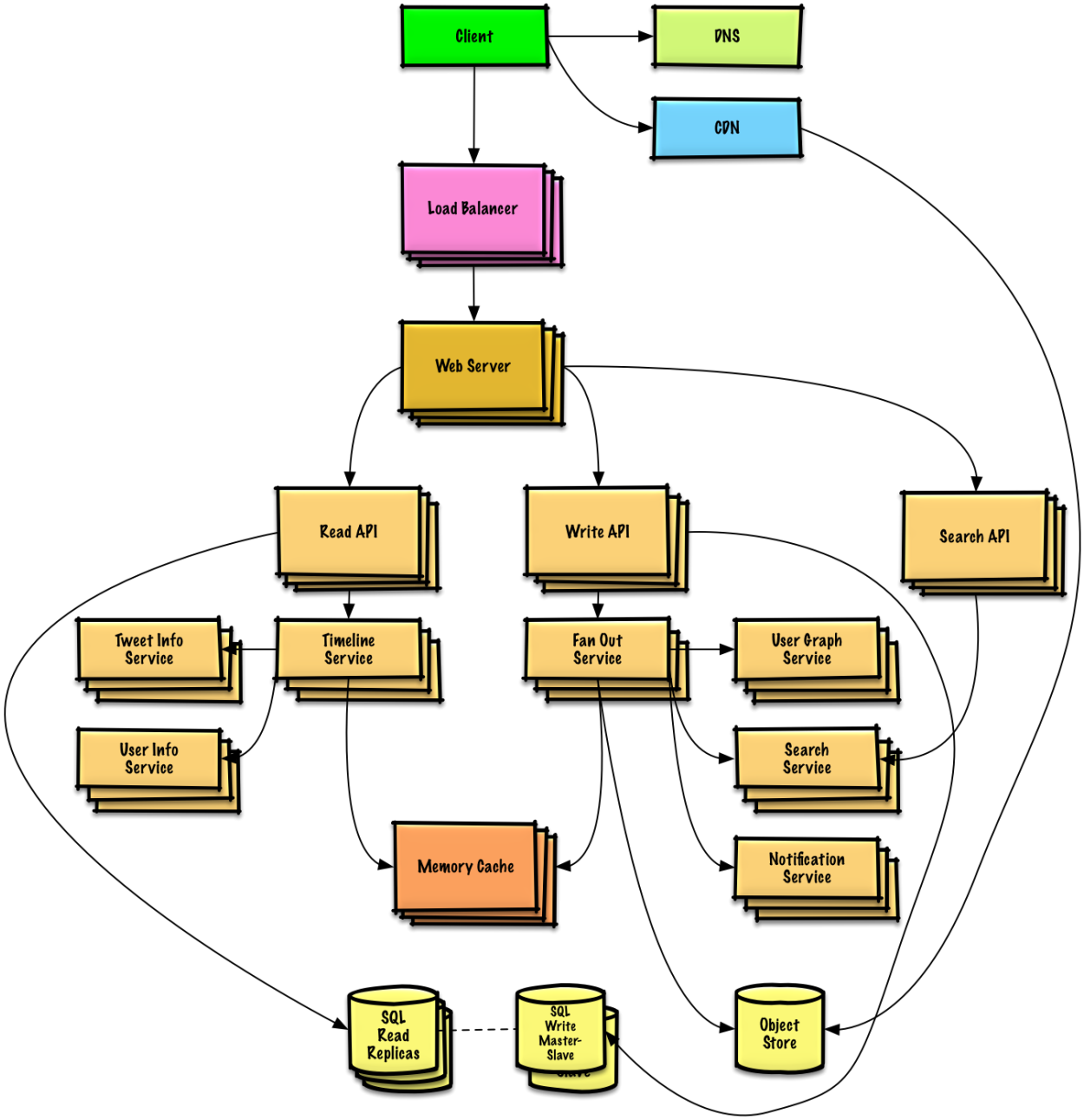
"How does Samsara ensure the alignment of its engineering teams with its cultural principles, especially as it scales globally?"

"What does success look like for a Senior Machine Learning Engineer at Samsara, and how is it measured?"

"Considering the hybrid nature of cloud-edge ML, how does Samsara manage challenges that arise in real-time data processing?"

"How do cross-functional teams at Samsara collaborate, especially when deploying models and ensuring optimal performance?"

"Are there opportunities for engineers to participate in academic or industry conferences to stay updated with the latest research?


Client
DNS
CDN
Load Balancer
Web Server
Read API
Write API
Search API
Tweet Info Service
Timeline Service
Fan Out Service
User Graph Service
User Info Service
Search Service
Memory Cache
Notification Service
SQL Read Replicas
SQL Write Master-Slave
Object Store

1. Problem Formulation

- Clarifying questions
- Use case(s) and business goal
- Requirements
    - Scope (features needed), scale, and personalization
    - Performance: prediction latency, scale of prediction
    - Constraints
    - Data: sources and availability
- Assumptions
- Translate an abstract problem into an ML problem
    - ML objective,
    - ML I/O,
    - ML category (e.g. binary classification, multi-classification, unsupervised learning, etc)
- Do we need ML to solve this problem?
    - Trade off between impact and cost
        - Costs: Data collection, data annotation, compute
        - if Yes, we choose an ML system to design. If No, follow a general system design flow.
        - Note: in an ML system design interview we can assume we need ML.

2. Metrics (Offline and Online)

- Offline metrics (e.g. classification, relevance metrics)
    - Classification metrics
        - Precision, Recall, F1, ROC AUC, P/R AUC, mAP, log-loss, etc
            - Imbalanced data
    - Retrieval and ranking metrics
        - Precision@k, Recall@k (do not consider ranking quality)
        - mAP, MRR, nDCG
    - Regression metrics: MSE, MAE,
    - Problem specific metrics
        - Language: BLEU, BLEURT, GLUE, ROUGE, etc
        - ads: CPE, etc
    - Latency
    - Computational cost (in particular for on-device)
- Online metrics

    - CTR
    - Task/session success/failure rate,
    - Task/session total (e.g. watch) times,
    - Engagement rate (like rate, comment rate)
    - Conversion rate
    - Revenue lift
    - Reciprocal rank of first click, etc,
    - Counter metrics: direct negative feedback (hide, report)
- Trade-offs b/w metrics

3. Architectural Components (MVP Logic)

- High level architecture and main components
    - Non-ML components:
        - user, app server, DBs, KGs, etc and their interactions
    - ML components:
        - Modeling modules (e.g. candidate generator, ranker, ect)
        - Train data generator

        ...
- Modular architecture design
    - Model 1 architecture (e.g. candidate generation)
    - Model 2 architecture (e.g. ranker, filter)
    - ...

4. Data Collection and Preparation

- Data needs
    - target variable
    - big actors in signals (e.g. users, items, etc)
    - type (e.g. image, text, video, etc) and volume
- Data Sources
    - availability and cost
    - implicit (logging), explicit (e.g. user survey)
- Data storage
- ML Data types
    - structured
        - numerical (discrete, continuous)
        - categorical (ordinal, nominal),
    - unstructured(e.g. image, text, video, audio)

- Labelling (for supervised)
    - Labeling methods
        - Natural labels (extracted from data e.g. clicks, likes, purchase, etc)
            - Missing negative labels (not clicking is not a negative label):
                - Negative sampling
        - Explicit user feedback
        - Human annotation (super costly, slow, privacy issues)
    - Handling lack of labels
    - Programmatic labeling methods (noisy, pros: cost, privacy, adaptive)
        - Semi-supervised methods (from an initial smaller set of labels e.g. perturbation based)
        - Weak supervision (encode heuristics e.g. keywords, regex, db, output of other ML models)
    - Transfer learning:
        - pre-train on cheap large data (e.g. GPT-3),
        - zero-shot or fine-tune for downstream task
    - Active learning
    - Labeling cost and trade-offs
- Data augmentation
- Data Generation Pipeline
    - Data collection/ingestion (offline, online)
    - Feature generation (next)
    - Feature transform
    - Label generation
    - Joiner

5. Feature Engineering

- Choosing features
    - Define big actors (e.g. user, item, document, query, ad, context),
    - Define actor specific features (current, historic)
        - Example user features: user profile, user history, user interests
        - Example text features: n-grams (uni,bi), intent, topic, frequency, length, embeddings
    - Define cross features (e.g. user-item, or query-document features)
        - Example query-document features: tf-idf
        - Example user-item features: user-video watch history, user search history, user-ad interactions(view, like)
    - Privacy constraints
- Feature representation

    - One hot encoding
    - Embeddings
        - e.g. for text, image, graphs, users (how), stores, etc
        - how to generate/learn?
        - pre-compute and store
    - Encoding categorical features (one hot, ordinal, count, etc)
    - Positional embeddings
    - Scaling/Normalization (for numerical features)
- Preprocessing features
    - Needed for unstructured data
        - Text: Tokenize (Normalize, pre-tokenize, tokenizer model (ch/word/subword level), post-process (add special tokens))
        - Images: Resize, normalize
        - Video: Decode frames, sample, resize, scale and normalize
- Missing Values
- Feature importance
- Featurizer (raw data -> features)
- Static (from feature store) vs dynamic (computed online) features

6. Model Development and Offline Evaluation

- Model selection (MVP)
    - Heuristics -> simple model -> more complex model -> ensemble of models
        - Pros and cons, and decision
        - Note: Always start as simple as possible (KISS) and iterate over
    - Typical modeling choices:
        - Logistic Regression
        - Decision tree variants
            - GBDT (XGBoost) and RF
        - Neural networks
            - FeedForward
            - CNN
            - RNN
            - Transformers
    - Decision Factors
        - Complexity of the task
        - Data: Type of data (structured, unstructured), amount of data, complexity of data
        - Training speed
        - Inference requirements: compute, latency, memory

        - Continual learning
        - Interpretability
    - Popular NN architectures
- Dataset
    - Sampling
        - Non-probabilistic sampling
        - Probabilistic sampling methods
            - random, stratified, reservoir, importance sampling
    - Data splits (train, dev, test)
        - Portions
        - Splitting time-correlated data (split by time)
            - seasonality, trend
        - Data leakage:
            - scale after split,
            - use only train split for stats, scaling, and missing vals
    - Class Imbalance
        - Resampling
        - weighted loss fcn
        - combining classes
- Model training
    - Loss functions
        - MSE, Binary/Categorical CE, MAE, Huber loss, Hinge loss, Contrastive loss, etc
    - Optimizers
        - SGD, AdaGrad, RMSProp, Adam, etc
    - Model training
        - Training from scratch or fine-tune
    - Model validation
    - Debugging
    - Offline vs online training
    - Model offline evaluation
    - Hyper parameter tuning
        - Grid search
    - Iterate over MVP model
        - Model Selection
        - Data augmentation
        - Model update frequency
    - Model calibration

7. Prediction Service

- Data processing and verification
- Web app and serving system
- Prediction service
- Batch vs Online prediction
    - Batch: periodic, pre-computed and stored, retrieved as needed - high throughput
    - Online: predict as request arrives - low latency
    - Hybrid: e.g. Netflix: batch for titles, online for rows
- Nearest Neighbor Service
    - Approximate NN
        - Tree based, LSH, Clustering based
- ML on the Edge (on-device AI)
    - Network connection/latency, privacy, cheap
    - Memory, compute power, energy constraints
    - Model Compression
        - Quantization
        - Pruning
        - Knowledge distillation
        - Factorization

8. Online Testing and Model Deployment

- A/B Experiments
    - How to A/B test?
        - what portion of users?
        - control and test groups
        - null hypothesis
- Bandits
- Shadow deployment
- Canary release

9. Scaling, Monitoring, and Updates

- Scaling for increased demand (same as in distributed systems)
    - Scaling general SW system (distributed servers, load balancer, sharding, replication, caching, etc)
        - Train data / KB partitioning
    - Scaling ML system
        - Distributed ML

            - Data parallelism (for training)
            - Model parallelism (for training, inference)
                - Asynchronous SGD
                - Synchronous SGD
        - Distributed training
            - Data parallel DT, RPC based DT
        - Scaling data collection
            - MT for 1000 languages
            - NLLB
        - Monitoring, failure tolerance, updating (below)
        - Auto ML (soft: HP tuning, hard: arch search (NAS))
- Monitoring:
    - Logging
        - Features, predictions, metrics, events
    - Monitoring metrics
        - SW system metrics
        - ML metrics (accuracy related, predictions, features)
            - Online and offline metric dashboards
    - Monitoring data distribution shifts
        - Types: Covariate, label and concept shifts
        - Detection (stats, hypothesis testing)
        - Correction
- System failures
    - SW system failure
        - dependency, deployment, hardware, downtime
    - ML system failure
        - data distribution difference (test vs online)
        - feedback loops
        - edge cases (e.g. invalid/junk input)
        - data distribution changes
    - Alarms
        - failures (data pipeline, training, deployment), low metrics, etc
- Updates: Continual training
    - Model updates
        - train from scratch or a base model
        - how often? daily, weekly, monthly, etc
    - Auto update models
    - Active learning
    - Human in the loop ML