EXAM GUIDE

Google Cloud Certified Professional Machine Learning Engineer Certification

A Professional Machine Learning Engineer designs, builds, and productionizes ML models to solve business challenges using Google Cloud technologies and knowledge of proven ML models and techniques. The ML Engineer considers responsible AI throughout the ML development process, and collaborates closely with other job roles to ensure long-term success of models. The ML Engineer should be proficient in all aspects of model architecture, data pipeline interaction, and metrics interpretation. The ML Engineer needs familiarity with foundational concepts of application development, infrastructure management, data engineering, and data governance. Through an understanding of training, retraining, deploying, scheduling, monitoring, and improving models, the ML Engineer designs and creates scalable solutions for optimal performance.

Exam Outline

Section 1: Framing ML problems

1.1 Translating business challenges into ML use cases. Considerations include:

- Choosing the best solution (ML vs. non-ML, custom vs. pre-packaged [e.g., AutoML, Vision API]) based on the business requirements
- Defining how the model output should be used to solve the business problem

- Deciding how incorrect results should be handled
- Identifying data sources (available vs. ideal)

1.2 Defining ML problems. Considerations include:

- Problem type (e.g., classification, regression, clustering)
- Outcome of model predictions
- Input (features) and predicted output format

1.3 Defining business success criteria. Considerations include:

- Alignment of ML success metrics to the business problem
- Key results
- Determining when a model is deemed unsuccessful

1.4 Identifying risks to feasibility of ML solutions. Considerations include:

- Assessing and communicating business impact
- Assessing ML solution readiness
- Assessing data readiness and potential limitations
- Aligning with Google's Responsible AI practices (e.g., different biases)

Section 2: Architecting ML solutions

2.1 Designing reliable, scalable, and highly available ML solutions. Considerations include:

- Choosing appropriate ML services for the use case (e.g., Cloud Build, Kubeflow)
- Component types (e.g., data collection, data management)
- Exploration/analysis
- Feature engineering
- Logging/management

- Automation
- Orchestration
- Monitoring
- Serving

2.2 Choosing appropriate Google Cloud hardware components. Considerations include:

- Evaluation of compute and accelerator options (e.g., CPU, GPU, TPU, edge devices)

2.3 Designing architecture that complies with security concerns across sectors/industries. Considerations include:

- Building secure ML systems (e.g., protecting against unintentional exploitation of data/model, hacking)
- Privacy implications of data usage and/or collection (e.g., handling sensitive data such as Personally Identifiable Information [PII] and Protected Health Information [PHI])

Section 3: Designing data preparation and processing systems

3.1 Exploring data (EDA). Considerations include:

- Visualization
- Statistical fundamentals at scale
- Evaluation of data quality and feasibility
- Establish data constraints (e.g., TFDV)

3.2 Building data pipelines. Considerations include:

- Organize and optimize training datasets
- Data validation
- Handling missing data

- Handling outliers
- Data leakage

3.3 Creating input features (feature engineering). Considerations include:

- Ensuring consistent data pre-processing between training and serving
- Encoding structured data types
- Feature selection
- Class imbalance
- Feature crosses
- Transformations (TensorFlow Transform)

Section 4: Developing ML models

4.1 Building models. Considerations include:

- Choice of framework and model
- Modeling techniques given interpretability requirements
- Transfer learning
- Data augmentation
- Semi-supervised learning
- Model generalization and strategies to handle overfitting and underfitting

4.2 Training models. Considerations include:

- Ingestion of various file types into training (e.g., CSV, JSON, IMG, parquet or databases, Hadoop/Spark)
- Training a model as a job in different environments
- Hyperparameter tuning
- Tracking metrics during training
- Retraining/redeployment evaluation

4.3 Testing models. Considerations include:

- Unit tests for model training and serving
- Model performance against baselines, simpler models, and across the time dimension
- Model explainability on AI Platform

4.4 Scaling model training and serving. Considerations include:

- Distributed training
- Scaling prediction service (e.g., AI Platform Prediction, containerized serving)

Section 5: Automating and orchestrating ML pipelines

5.1 Designing and implementing training pipelines. Considerations include:

- Identification of components, parameters, triggers, and compute needs (e.g., Cloud Build, Cloud Run)
- Orchestration framework (e.g., Kubeflow Pipelines/AI Platform Pipelines, Cloud Composer/Apache Airflow)
- Hybrid or multi-cloud strategies
- System design with TFX components/Kubeflow DSL

5.2 Implementing serving pipelines. Considerations include:

- Serving (online, batch, caching)
- Google Cloud serving options
- Testing for target performance
- Configuring trigger and pipeline schedules

5.3 Tracking and auditing metadata. Considerations include:

- Organizing and tracking experiments and pipeline runs

- Hooking into model and dataset versioning
- Model/dataset lineage

Section 6: Monitoring, optimizing, and maintaining ML solutions

6.1 Monitoring and troubleshooting ML solutions. Considerations include:

- Performance and business quality of ML model predictions
- Logging strategies
- Establishing continuous evaluation metrics (e.g., evaluation of drift or bias)
- Understanding Google Cloud permissions model
- Identification of appropriate retraining policy
- Common training and serving errors (TensorFlow)
- ML model failure and resulting biases

6.2 Tuning performance of ML solutions for training and serving in production. Considerations include:

- Optimization and simplification of input pipeline for training
- Simplification techniques