**Rubric for Model Deployment Final Assignment**

**Course**: Model Deployment | **Target**: Third-year HBO IT students

**Deliverable for the course**: A report detailing the deployment strategy and implementation of an AI model for your project, demonstrating server-side or client-side deployment, evaluation, and research into user requirements and theoretical foundations.

**Passing Criteria**: Weighted average ≥55%, no 'U' marks, maximum one 'D' mark.

What is the Deliverable?

* Implementation and deployment of the application itself, application here means the interface between a user/client and the “model”
* A report that provides an overview of the application and deployment strategy

**A brief report (1-5 pages), that contains:**

1. **Short introduction of the use-case | Needs & Requirements**
   * In which ways might the client/user want to use your model?
   * What factors are most important when it comes to functionalities, robustness and performance?
2. **Brief description of your model & data?**
   * Outline of the model and data characteristics that are relevant for deployment
   * Description of Input & Output Data | Structured vs. unstructured, size of data
   * Computational demands of the model
3. **A description of your deployment strategy:**
   * Which choices did you make for the deployment?
   * Describe the solutions used for the final implementation
   * Motivate that based on first two points: How did you align the user needs with technical requirements?
   * How can the model be monitored during usage? (logging, monitoring)
   * Describe how you approached the lifecycle-management of the application:
     + How did you implement version-control of code, model, and data?
     + What steps are needed for shipping a new version of your application?
     + How do you test and validate the main functionalities before deploying?
4. **Documentation of your implementation of the application (will be further defined):**
   * **Additionallly, provide**
   * Link to the repository with a short explanation.
   * Screenshots that showcase both the implementation but also aspect like monitoring and error-handling.

**Grading criteria:**

**1. Deployment Implementation (Weight: 3x)**

**Goal:** Deliver a working, scalable deployment of the application.

**Passing criteria:** The application must expose a functional interface to at least one of your models. The interface can be either as a web-based UI or a callable HTTP endpoint.

**Next to that, the other criteria are:**

* Robustness (Basic error handling, failure status codes, uptime checks, etc., access control)
* Usability (Provide logical gateways to your model - e.g. filtering data, adjusting parameters, retraining)
* Performance & Scalability (Latency, reliability)

***The most important aspect here is that you implement a solution that fulfills the requirements of your use-case. E.g. if one of the criteria listed above is***

***Important:*** If you choose a web-based UI, please justify that in the deployment strategy.

**2. Model Lifecycle Management (Weight: 2x)**

Goal: The implementation of your application should support the lifecycle of the model.

**Passing Criteria: You application includes a basic implementation of version control for code| model | data**

**Next to that, other criteria are:**

* **Monitoring & Logging:** You generate insights and documenation while your application is running
* **Reproducibility & Standardization for** (e.g. automated workflow): There is a clear order of steps for redeployment that can be triggered with a limited number of actions. There shouldn’t be a strong dependency on a local environment.
* **Validation & Testing before redeployment:** You ensure that both interface and model function properly before re-deploying.

**3. Requirements Analysis and Deployment Strategy (Weight: 2x)**

**Goal:** The deployment should reflect a clear understanding of both user-facing requirements and technical constraints. Your choices should be grounded in a deliberate, well-argued strategy.

**Passing Criteria:** You identify relevant requirements from the user side, understand key model and data constraints, and describe how your deployment addresses both. You outline your main architectural choices and justify them based on this analysis.

**Next to that, the other criteria are:**

* **User-Side Requirements:** You describe what users, front-ends, or connected systems need. This may include:
  + Speed or responsiveness
  + Interactivity (UI vs. API)
  + Security or authentication
  + Output format and consistency
  + Integration points with other tools
* **Model & Data Constraints:** You describe technical limitations or requirements of your model or dataset, such as:
  + Model size and runtime behavior (does it take a lot of resources to run your model?)
  + Model weights & data:
* **Fit & Trade-offs:** You clearly relate the above requirements to your final implementation. You identify:
  + Where user needs and technical limits align
  + Where compromises had to be made
  + Whether the solution is appropriately scoped (not over/under-engineered)
* **Deployment Strategy:** You describe the key components of your deployment stack, and explain:
  + Why did you choose this specific stack (e.g. FastAPI + Docker + Azure Container instances)
  + What the main strengths and weaknesses of this choice are
  + How your choices support future development or maintenance

**What is a fail?**

* The access to the model via interface does not work at all, or there are major flaws. (e.g. no error handling at all, application keeps crashing, only running on localhost)
* There is no documentation or version control at all (for either code or model)
* There are major logical flaws in the report

**Example: What is the Bare Minimum Application? (passing grade: 5.5)**

A Flask app exposes a model via a single HTTP endpoint, providing predictions based on inputs. Error handling is existent but could be more specific or is partially incorrect, e.g. providing 500 as an error sometimes for invalid requests. The logs are visible via automatic Flask logging and displayed in the console. (e.g. Azure container app)

There is basic version control of the code and multiple versions of the model available. However, there is significant room for improvement, and it is not a sustainable solution, the absolute minimum would be something like model\_weights\_v1.h5, model\_weights\_v2.h5, and a git repo with more than the initial commit.

The report lists the steps towards deploying a new version, but they contain a larger number of manual steps and a larger effort to replicate.

**Example: What is a Great Application? (best grade: 10)**

This application provides a **robust, scalable, and maintainable deployment** of a resume-screening model via a well-structured API, accessible through secure HTTP endpoints. The system is deployed via **Azure Web aPP**, with automated CI/CD pipelines and integrated logging and monitoring.

**1. Deployment Implementation**

* **Interface:** The model is served through a REST API that provides both /predict, /health, and /metadata. Each endpoint has clear schema validation (e.g. using pydantic) and returns structured error responses (e.g., 422 for invalid input, 500 only for internal errors).
* **Robustness:** Implements exception handling with meaningful status codes and logging. Includes uptime monitoring via Azure and automated health checks.
* **Performance & Scalability:** Deployed within a scalable solution, build-time and latency are reduced by design choices - e.g. storing pre-computed data in storage solution and loading during building.

**2. Model Lifecycle Management**

* **Version Control:** Code and model versions are tracked via Git and Models are stored in Azure Blob Storage with clear version tags (model-v1.2.0.pt), and deployment references are documented. (version of code &  
   model)
* **Reproducibility:** The application supports redeployment using a Makefile (make deploy) (or any kind of script that goes through distinct steps of building, testing, and deploying) - bash scripts, powershell scripts, GitHub Actions workflow that builds the image, runs unit and integration tests, and deploys to Azure on push to main. No manual setup is needed beyond setting environment variables.
* **Validation:** A suite of pytest-based tests covers API contracts and model outputs. Each deployment requires successful test runs on GitHub Actions before the container is updated.

**3. Requirements Analysis and Deployment Strategy**

* **User Needs:** The deployment strategy is aligned with specific performance, reliability, and integration requirements. This includes latency targets, response formats, and security constraints.
* **Model Constraints:** The model’s computational footprint, runtime, and data sensitivity are factored into infrastructure decisions.
* **Technology Fit:** The tech stack is selected based on clarity, maintainability, and suitability—not trendiness. The trade-offs (e.g. speed vs. cost, simplicity vs. flexibility) are explicitly considered.
* **Scalability and Extensibility:** The system supports updates, rollback, and gradual improvements—such as comparing model versions or enriching metadata—without reengineering the core pipeline.