Ready Tensor Repository Assessment Framework

Date: March 2025

criteria_category	sub_category	criteria	description	essential	professional	elite
Documentation	Core Documentation	README File Exists	Repository has a README file in the root directory.	yes	yes	yes
Documentation	Core Documentation	Descriptive Project Title	README has a clear, descriptive title near the top that accurately represents the project's purpose and content. The title should be concise yet informative, avoiding uncommon acronyms or jargon without explanation.	yes	yes	yes
Documentation	Core Documentation	Concise Project Summary	Clear statement of project or concise project summary near top of README	yes	yes	yes
Documentation	Core Documentation	Target Audience Defined	Clear identification of the target audience who can benefit from this project	no	yes	yes
Documentation	Core Documentation	Complete Project Description	Clear explanation of the project that provides enough context to understand its functionality, approach, and value. Typically 1-2 paragraphs that expand on the purpose statement with more details.	yes	yes	yes
Documentation	Core Documentation	Well-Structured README	Well-structured readme with clear headings and logical organization of information into distinct sections such as overview, installation, usage, etc.	yes	yes	yes

Documentation	Repository Structure Documentation	Main Directory Purpose Overview	Explanation of at least the main directories and usage scripts (such as /src, /data, /models, etc.).	yes	no	no
Documentation	Repository Structure Documentation	Full Directory Structure Documentation	Comprehensive explanation of all directories and files	no	yes	yes
Documentation	Setup Documentation	Prerequisites Clearly Stated	Lists the necessary knowledge, hardware requirements such as GPU, and system compatibility information before installation. This does not include software dependencies (e.g., Python libraries), which are covered under Environment & Dependency Management.	no	yes	yes
Documentation	Setup Documentation	Basic Installation Guide Present	Basic installation information (dependencies)	yes	no	no
Documentation	Setup Documentation	Detailed Installation Instructions	Detailed installation instructions for repo and prerequisites	no	yes	yes
Documentation	Setup Documentation	Environment Setup Requirements Specified	Specifies required Python version, dependency management (e.g., requirements.txt, poetry.lock, conda.yaml), and recommendations for environment reproducibility (e.g., Docker, Conda, virtual environments). Does not cover hardware requirements or system compatibility, which are listed under Prerequisites Clearly Stated.	no	yes	yes
Documentation	Usage Documentation	Basic Usage Instructions Present	Basic information on how to use the repo (main script and execution)	yes	no	no

Documentation	Usage Documentation	Step-by-Step Usage Guide Present	Detailed step-by-step usage instructions (data prep, execution, outputs)	no	yes	yes
Documentation	Usage Documentation	Practical Code Examples Included	Concrete, executable code examples that demonstrate key functionality in a clear, concise manner.	no	yes	yes
Documentation	Usage Documentation	Testing Instructions Provided	Instructions for running tests	no	yes	yes
Documentation	Technical Documentation	Data Requirements Specified	Documentation of expected data formats and setup	no	yes	yes
Documentation	Technical Documentation	Methodology Documentation Provided	Basic methodology information or reference to external documentation	no	yes	yes
Documentation	Technical Documentation	Key Parameters Explained	Documentation of key modeling parameters and considerations	no	yes	yes
Documentation	Technical Documentation	Configuration Options Explained	Information on key configuration options	no	yes	yes
Documentation	License Identification	License Identified	README includes a clear mention of the project's license(s).	yes	yes	yes
Documentation	Community Documentation	Contribution Guidelines Provided	Guidelines for project contribution	no	yes	yes
Documentation	Community Documentation	Change History Documented	Repository includes changelog information either in README or in a dedicated file (like CHANGELOG.md), documenting significant version changes.	no	no	yes
Documentation	Community Documentation	Maintainer Contact Information Provided	Clear contact information for maintainers	no	no	yes

Repository Structure	Core Organization	Basic Modular Organization	Repository has basic organizational structure with some logical separation of files (not all files in root directory)	yes	no	no
Repository Structure	Core Organization	Code Separation	Modeling/application code is organized in dedicated module structure (e.g., src/ directory with submodules)	no	yes	yes
Repository Structure	Core Organization	Data Separation	Data is organized in dedicated directory (e.g., /data, /inputs, etc.)	no	yes	yes
Repository Structure	Core Organization	Config Separation	Configuration is properly separated from code in dedicated files/directories	no	yes	yes
Repository Structure	Core Organization	Organized Notebooks	If Jupyter notebooks are present, they are organized in a logical manner (e.g., in dedicated directories) with names that reflect their purpose and sequence (e.g., "01_data_exploration.ipynb", "02_feature_engineering.ipynb")	no	yes	yes
Repository Structure	Core Organization	Documentation and References Separation	Documents and external materials such as pdfs, papers, etc. are properly organized in dedicated locations (e.g., docs/ directory)	no	yes	yes
Repository Structure	Core Organization	Asset Organization	Non-code assets (images, models, etc.) are organized in dedicated directories with clear purpose	no	yes	yes
Repository Structure	Core Organization	Test Directory Organized	Tests are organized in a dedicated structure (e.g., tests/ directory mirroring the main code structure)	no	yes	yes
Repository Structure	Core Organization	Logical Repository Root	Repository root is clean with only essential files (README, license, .gitignore, setup files) with detailed content in subdirectories	no	yes	yes

Repository Structure	Core Organization	Appropriate .gitignore	Repository includes a .gitignore file appropriate for the project type/language	yes	yes	yes
Repository Structure	Naming Practices	Consistent File and Dir Naming Convention	Files and directories follow a single, consistent naming convention (snake_case, camelCase, etc.)	yes	yes	yes
Repository Structure	Naming Practices	Descriptive File and Dir Naming	Files and directories have descriptive names that clearly indicate their purpose or content (no generic names like notebook1.ipynb)	yes	yes	yes
Repository Structure	Naming Practices	Unambiguous Related Item Naming	Related files and directories use a consistent, clear naming scheme that avoids ambiguity (e.g., avoid "experiment.py", "experiment_new.py", "experiment_final.py" or "temp_models/", "models_new/", "final_models/")	yes	yes	yes
Repository Structure	Directory Design	Appropriate Directory Density	Directories contain a reasonable number of files and dirs (should be under 15 directories and files in single directories); does not apply to data directories	no	yes	yes
Repository Structure	Directory Design	Reasonable Directory Depth	Repository structure avoids excessive nesting of directories (typically no more than 5 levels deep)	no	yes	yes
Repository Structure	Directory Design	Clear Entry Points	Main execution entry points are clearly identified (e.g., main.py, app.py, or documentation pointing to them)	yes	yes	yes
Repository Structure	Environment & Dependencies	Environment Configuration Isolation	Environment-specific configuration files (.env, .env.example, .env.template, .gitconfig, docker-compose.yml, etc.) are properly placed and organized	no	yes	yes

Repository Structure	Environment & Dependencies	Organized Dependency Management	Package dependency files (requirements.txt, pyproject.toml, etc.) are properly placed at repository root or in designated dependency directories. Multiple dependency files (if present) follow logical organization (e.g., base_requirements.txt, dev_requirements.txt).	no	yes	yes
Environment and Dependencies	Dependency Management	Dependencies Listed	Repository clearly lists all project dependencies in standard formats (requirements.txt, setup.py, pyproject.toml, etc.).	yes	yes	yes
Environment and Dependencies	Dependency Management	Pinned Dependencies	Dependencies have specific versions to ensure reproducibility (preferably production dependencies pinned).	no	yes	yes
Environment and Dependencies	Dependency Management	Dependency Groups	Dependencies organized into logical groups (core, dev, test) via separate requirement files (e.g., requirements-dev.txt) or in configuration files (e.g., pyproject.toml, setup.py extras_require).	no	yes	yes
Environment and Dependencies	Environment Configuration	Python Version Specified	Required Python version(s) specified in configuration files like pyproject.toml, setup.py, runtime.txt, or .python-version.	no	yes	yes
Environment and Dependencies	Environment Configuration	Environment Managed	Repository contains configuration for virtual environments such as environment.yml (conda), Pipfile (pipenv), poetry.lock (poetry), .venv configurations, or similar environment management files.	no	yes	yes
Environment and Dependencies	Environment Configuration	Reproducible Environment	Lockfiles or exact environment specifications provided (poetry.lock, conda-lock, etc.).	no	no	yes

Environment and Dependencies	Environment Configuration	GPU Requirements Documented	GPU-specific dependencies or requirements indicated in configuration files (e.g., tensorflow-gpu in requirements.txt, CUDA versions in environment files, GPU constraints in Dockerfiles).	no	no	yes
Environment and Dependencies	Advanced Configuration	Containerization	Provides Dockerfile or equivalent containerization.	no	no	yes
License and Legal	License	License Presence	Repository includes a recognized license file (LICENSE, LICENSE.md, LICENSE.txt) in the root directory, explicitly stating the terms of use, modification, and distribution. If no separate file exists, the README or documentation must contain clear licensing terms.	yes	yes	yes
License and Legal	License	License Appropriateness	The repository's chosen license is suitable for its purpose, dependencies, and intended use, ensuring clarity on permissions and restrictions. Repositories with unclear or conflicting licenses do not meet this criterion.	yes	yes	yes
License and Legal	Rights and Permissions	Copyright Notice Presence	The repository includes explicit copyright statements in source files and documentation, indicating ownership and rights. This helps prevent ambiguity in legal rights and attribution.	no	no	yes

License and Legal	Rights and Permissions	Data Usage Rights Mentioned	For repositories handling datasets, clear documentation must state data ownership, licensing, usage rights, compliance requirements, and any restrictions.	no	yes	yes
License and Legal	Rights and Permissions	Model Usage Rights Mentioned	If the repository includes or references ML models, documentation must specify model ownership, licensing, usage terms, and redistribution policies.	no	yes	yes
License and Legal	Community	Code of Conduct Mentioned	The repository includes a Code of Conduct, outlining contributor behavior expectations, enforcement mechanisms, and reporting guidelines to foster an inclusive and respectful environment.	no	no	yes
Code Quality	General Structure	Modular Code Organization	Code organized into functions/methods rather than monolithic scripts	Yes	yes	yes
Code Quality	General Structure	Script Length Control	Individual scripts/modules have reasonable length (< 500 lines)	Yes	yes	yes
Code Quality	General Structure	Function Length Control	Functions and methods are reasonably sized (< 50 lines)	no	yes	yes
Code Quality	General Structure	Minimal Code Duplication	Limited code duplication (< 10% duplicate code)	no	yes	yes
Code Quality	Configuration	Configuration File Presence	Dedicated configuration files exist (e.g., config.py, config.json, .env)	Yes	yes	yes
Code Quality	Configuration	None or Limited Hardcoded Values	Limited hardcoded constants in core code	no	yes	yes
Code Quality	Configuration	Environment Variable Usage	References to environment variables for sensitive configurations	no	yes	yes

Code Quality	Logging	Logging Usage	Evidence of logging library usage	no	yes	yes
Code Quality	Logging	Logging Configuration Usage	Presence of logging configuration (levels, formats)	no	no	yes
Code Quality	Error Handling	Exception Usage	Presence of try/except blocks in the codebase	Yes	yes	yes
Code Quality	Error Handling	Custom Exception Classes	Definition of project-specific exception classes	no	no	yes
Code Quality	Testing	Test File Presence	Existence of test files (test_*.py or *_test.py)	no	yes	yes
Code Quality	Testing	Test Coverage	Test coverage metrics available (e.g., .coverage file)	no	no	yes
Code Quality	Testing	Test Framework Usage	Evidence of test framework (pytest, unittest)	no	yes	yes
Code Quality	Documentation	Docstring Presence	Functions, classes, modules have docstrings	no	yes	yes
Code Quality	Documentation	Docstring Completeness	Docstrings include parameters and return sections	no	yes	yes
Code Quality	Documentation	Type Hint Usage	Presence of type hints in function signatures	no	yes	yes
Code Quality	Style	Style Checker Configuration	Presence of style checker configuration files	no	yes	yes
Code Quality	Style	Consistent Formatting	Evidence of consistent indentation and formatting	no	yes	yes
Code Quality	Complexity	Cyclomatic Complexity	Functions maintain reasonable cyclomatic complexity (< 10)	no	yes	yes
Code Quality	Complexity	Class Size Control	Classes have reasonable number of methods (< 20)	no	yes	yes

Code Quality	AI/ML Specific	Random Seed Setting	Explicit random seed setting in code	Yes	yes	yes
Code Quality	AI/ML Specific	Data Validation Code	Input data validation logic present	no	yes	yes
Code Quality	AI/ML Specific	Model File Organization	ML models in dedicated modules/classes	no	yes	yes
Code Quality	Notebook Specific	Notebook Code Structure	Cells organized with fewer than 100 lines per cell	Yes	yes	yes
Code Quality	Notebook Specific	Notebook Documentation	Notebooks include markdown cells (>10% of cells)	Yes	yes	yes
Code Quality	Notebook Specific	Notebook Imports Evidence	Evidence of importing custom modules (not just standard libraries)	no	yes	yes
Code Quality	Notebook Specific	Clean Notebook Outputs	Either consistently cleared outputs or only relevant outputs retained	no	yes	yes
Code Quality	Package Structure	Package Organization	Proper package structure with init.py files	no	yes	yes