(plugin-system)= # vLLM's Plugin System The community frequently requests the ability to extend vLLM with custom features. To facilitate this, vLLM includes a plugin system that allows users to add custom features without modifying the vLLM codebase. This document explains how plugins work in vLLM and how to create a plugin for vLLM. ## How Plugins Work in vLLM Plugins are user-registered code that vLLM executes. Given vLLM's architecture (see [](#arch-overview)), multiple processes may be involved, especially when using distributed inference with various parallelism techniques. To enable plugins successfully, every process created by vLLM needs to load the plugin. This is done by the [load_general_plugins](https://github.com/vllm-project/vllm/blob/c76ac49d266e27aa3fea84ef2df1f813d24c91c7/vllm/plugins/__init__.py#L16) function in the `vllm.plugins` module. This function is called for every process created by vLLM before it starts any work. ## How vLLM Discovers Plugins vLLM's plugin system uses the standard Python `entry_points` mechanism. This mechanism allows developers to register functions in their Python packages for use by other packages. An example of a plugin: ```python # inside `setup.py` file from setuptools import setup setup(name='vllm_add_dummy_model', version='0.1', packages=['vllm_add_dummy_model'], entry_points={ 'vllm.general_plugins': ["register_dummy_model = vllm_add_dummy_model:register"] }) # inside `vllm_add_dummy_model.py` file def register(): from vllm import ModelRegistry if "MyLlava" not in ModelRegistry.get_supported_archs(): ModelRegistry.register_model("MyLlava", "vllm_add_dummy_model.my_llava:MyLlava") ``` For more information on adding entry points to your package, please check the [official documentation](https://setuptools.pypa.io/en/latest/userguide/entry_point.html). Every plugin has three parts: 1\. **Plugin group**: The name of the entry point group. vLLM uses the entry point group `vllm.general_plugins` to register general plugins. This is the key of `entry_points` in the `setup.py` file. Always use `vllm.general_plugins` for vLLM's general plugins. 2\. **Plugin name**: The name of the plugin. This is the value in the dictionary of the `entry_points` dictionary. In the example above, the plugin name is `register_dummy_model`. Plugins can be filtered by their names using the `VLLM_PLUGINS` environment variable. To load only a specific plugin, set `VLLM_PLUGINS` to the plugin name. 3\. **Plugin value**: The fully qualified name of the function to register in the plugin system. In the example above, the plugin value is `vllm_add_dummy_model:register`, which refers to a function named `register` in the `vllm_add_dummy_model` module. ## Types of supported plugins \- **General plugins** (with group name `vllm.general_plugins`): The primary use case for these plugins is to register custom, out-of-the-tree models into vLLM. This is done by calling `ModelRegistry.register_model` to register the model inside the plugin function. \- **Platform plugins** (with group name `vllm.platform_plugins`): The primary use case for these plugins is to register custom, out-of-the-tree platforms into vLLM. The plugin function should return `None` when the platform is not supported in the current environment, or the platform class's fully qualified name when the platform is supported. ## Guidelines for Writing Plugins \- **Being re-entrant**: The function specified in the entry point should be re-entrant, meaning it can be called multiple times without causing issues. This is necessary because the function might be called multiple times in some processes. ## Compatibility Guarantee vLLM guarantees the interface of documented plugins, such as `ModelRegistry.register_model`, will always be available for plugins to register models. However, it is the responsibility of plugin developers to ensure their plugins are compatible with the version of vLLM they are targeting. For example,

`"vllm_add_dummy_model.my_llava:MyLlava"` should be compatible with the

version of vLLM that the plugin targets. The interface for the model may

change during vLLM's development.