RKNN3 C API Reference Manual

Document ID: RK-YH-YF-425

Version: V1.0.0

Date: 2025-12-22

Confidentiality: □Top Secret □Secret □Internal ■Public

DISCLAIMER

Trademark Statement

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian,PRC

Website: www.rock-chips.com

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Preface

Overview

This document is the Rockchip RKNN3 C API Reference Manual.

Target Audience

This document (this guide) is primarily intended for the following engineers:

- Technical Support Engineers
- Software Development Engineers

Revision History

Version	Author	Date	Description	Approved By
V0.1.0	HPC Team	2025-05-12	Initial version	Vincent
V0.2.0	HPC Team	2025-08-04	Updated API interface descriptions	Vincent
V0.3.0b0	HPC Team	2025-09-10	Added description for rknn3_set_internal_mem interface; added KV cache storage methods and data type definitions; improved LLM configuration struct content	Vincent
V0.4.0b0	HPC Team	2025-11-03	Update the rknn3_vocab_info and rknn3_llm_multimodal_tensor structures and related interface descriptions.	Vincent
V0.5.0	HPC Team	2025-11-29	Added post-processing related query commands and a custom operator plugin mechanism; adjusted the output callback function interface; added YUV input related structures and interface descriptions.	Vincent
V1.0.0	HPC Team	2025-12-22	Add performance and memory analysis interfaces.	Vincent

Table of Contents

1. Overview

The RKNN3 C API is the C language interface for the RKNN3 Runtime. Developers use C/C++ to develop applications and deploy model inference through the RKNN3 C API. This document describes the functions, data structures, status codes, and other definitions of the RKNN3 C API.

2. Hardware Platforms

This document applies to the following hardware platform:

- RK1820
- RK1828

3. RKNN3 API Compilation Notes

In the RKNN3 SDK, the application compilation and execution are controlled by the host SoC, with the coprocessor serving only as an NPU acceleration unit. The host SoC connects to the coprocessor via a high-speed PCIe/USB interface, such as an RK3588 connected to an RK1820/RK1828.

When compiling an RKNN3 application, developers need to include the rknn3_api.h header file and link with the librknn3_api.so runtime library. rknn3_api.h defines basic inference interfaces for general deep learning model deployment (e.g., rknn3_init, rknn3_run) as well as a series of session interfaces specifically optimized for large language models (e.g., rknn3_session_init, rknn3_session_run). These interfaces provide functionalities like KV cache management, LoRA support, and sampling strategy configuration. Developers need to select the appropriate cross-compilation toolchain based on the target host's hardware platform and system type and ensure that the RKNN3 API library for the corresponding platform is correctly linked.

For instructions on downloading and installing the cross-compiler for the host SoC, please refer to the <`Rockchip_RK182X_Quick_Start_RKNN3_SDK`> document.

4. RKNN3 C API Call Flow

4.1 Non-Large Language Model Inference Flow

The inference flow for non-large language models like CNNs/ViTs using the basic API mainly includes the following steps. First, the user needs to call the rknn3_init interface to initialize the runtime context, and then load the model using rknn3_load_model_from_path or rknn3_load_model_from_data. After the model is loaded, rknn3_model_init must be called for model initialization and configuration. Subsequently, the user can query the attribute information of input and output tensors through the rknn3_query interface and allocate memory for inputs and outputs. When inference, the rknn3_run interface is called to perform inference. Finally, rknn3_destroy is used to release the context resources. The inference flow for the basic API is shown in Figure 4-1.

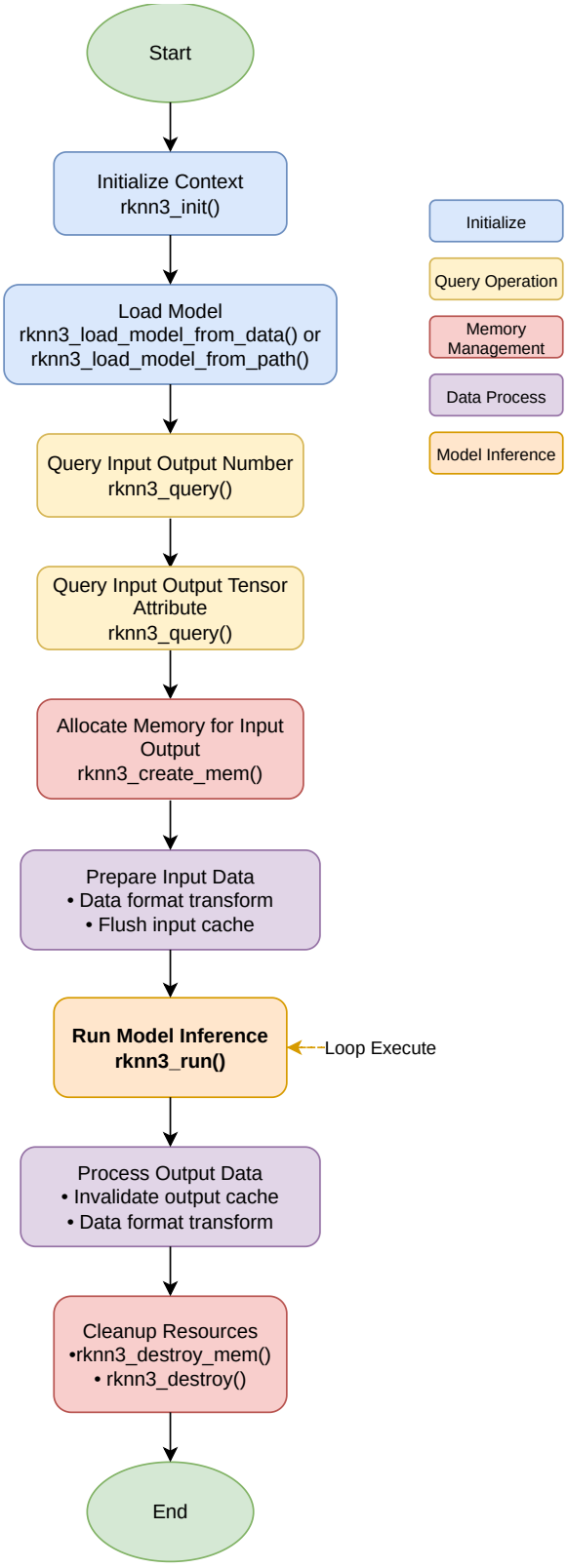

Figure 4-1. Basic API Inference Flow

4.2 Large Language Model Inference Flow

The RKNN3 large language model inference flow typically includes the following main steps. First, the user needs to get a list of available devices by calling rknn3_find_devices, initialize the context with rknn3_init, load the model file using rknn3_load_model_from_path, and complete the model initialization configuration with rknn3_model_init. Next, the user needs to prepare resources such as the tokenizer and embeddings and set up the rknn3_llm_param structure to configure session parameters. Then, call rknn3_session_init to initialize the session and set callback functions for the inference process via rknn3_session_set_callback. Before inference, a chat template can be set or the KV cache can be cleared as needed.

During the inference phase, the user constructs an rknn3_llm_input structure and calls the rknn3_session_run interface for inference. The inference results are obtained asynchronously through callback functions. After inference is complete, performance data can be collected, or multi-turn dialogues can be conducted. Finally, the rknn3_session_destroy and rknn3_destroy interfaces must be called to release session and context resources, completing the entire inference flow. The large language model inference flow is shown in Figure 4-2.

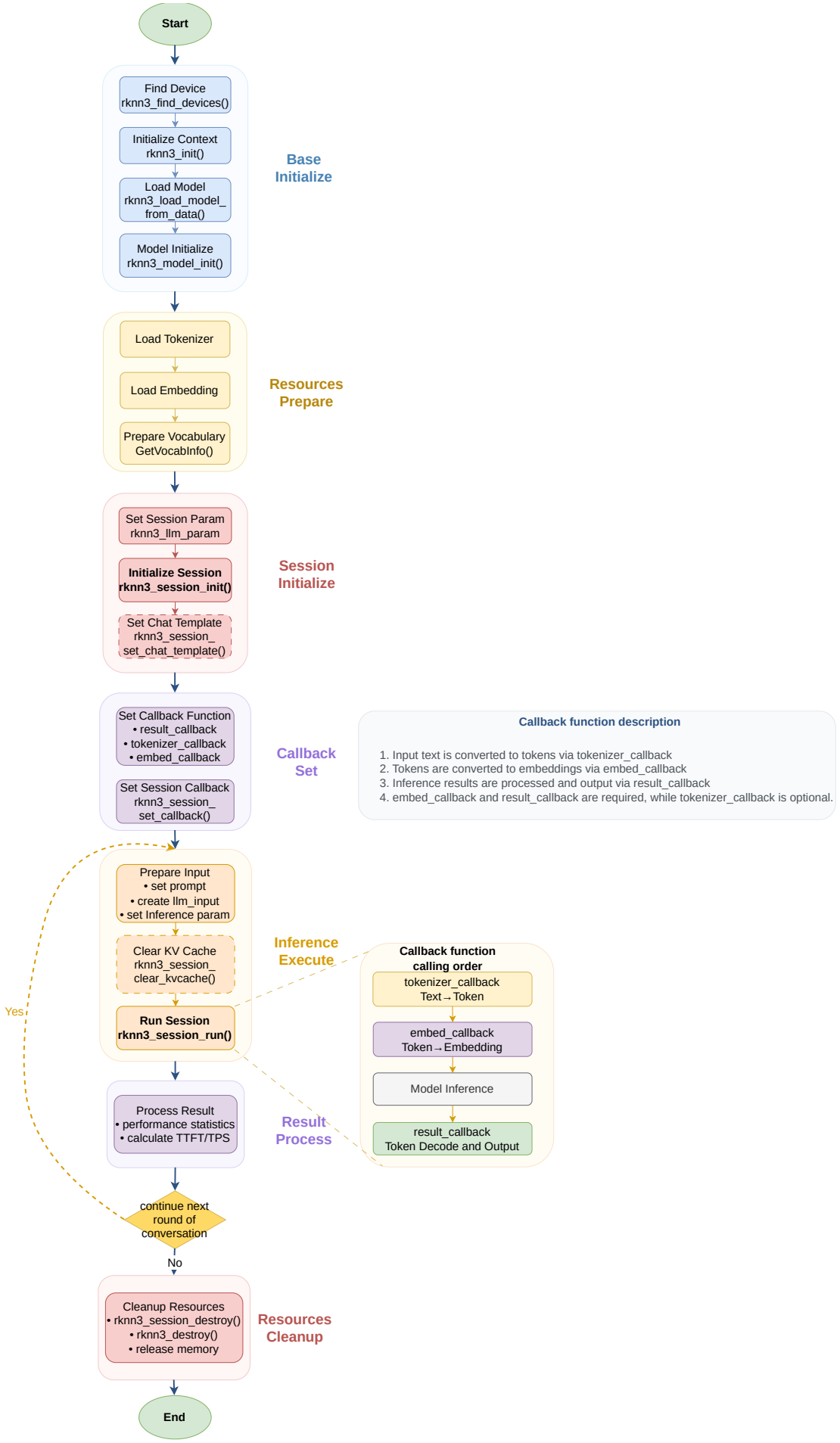

Figure 4-2. Large Language Model Inference Flow

5. RKNN3 C API Description

5.1. API Functions

5.1.1. rknn3_get_type_string

Fetches the string representation of a tensor type.

API	rknn3_get_type_string
Function	Gets the string representation of a tensor type.
Parameters	rknn3_tensor_type type: The type of the tensor.
Return	const char* The string for the type.

5.1.2. rknn3_get_qnt_type_string

Fetches the string representation of a quantization type.

API	rknn3_get_qnt_type_string
Function	Gets the string representation of a quantization type.
Parameters	rknn3_tensor_qnt_type type: The quantization type of the tensor.
Return	const char* The string for the quantization type.

5.1.3. rknn3_get_layout_string

Fetches the string representation of a layout type.

API	rknn3_get_layout_string
Function	Gets the string representation of a layout type.
Parameters	rknn3_tensor_layout layout: The layout type of the tensor.
Return	const char* The string for the layout type.

5.1.4. rknn3_init

This interface is used to initialize the RKNN3 runtime context and must be called before loading and running a model. context is an output parameter, and init_extend can be used to specify extended information such as device ID. After initialization, rknn3_destroy must be called to release resources when they are no longer needed. If initializing multiple times, ensure that resources are properly released to avoid memory leaks.

API	**rknn3_init**
Function	Initializes a new RKNN3 runtime context for running RKNN3 models on Rockchip NPU hardware.
Parameters	rknn3_context* context: Pointer to the RKNN3 context handle to be initialized.
	rknn3_init_extend* init_extend: Pointer to device-specific initialization information.
Return	int Status code: see 5.2. Status Codes.
Note	After use, please call rknn3_destroy(context) to release the context resources.

5.1.5. rknn3_load_model_from_path

This interface loads an RKNN3 model into an initialized context from a specified model file path and weight file path. model_path and weight_path are the paths to the model and weight files, respectively; weight_path can be NULL. Before calling, ensure that the context has been initialized and the paths are valid. An error code is returned on failure. After successful loading, model initialization and inference can be performed.

API	**rknn3_load_model_from_path**
Function	Loads an RKNN3 model from a file path into the specified context.
Parameters	rknn3_context context: The RKNN3 context handle.
	const char* model_path: The path to the RKNN3 model file.
	const char* weight_path: The path to the RKNN3 weight file.
Return	int Status code: see 5.2. Status Codes.

5.1.6. rknn3_load_model_from_data

This interface loads an RKNN3 model from memory data, suitable for scenarios where the model and weights are already in memory. model_data and weight_data are memory pointers to the model and weights, and model_size and weight_size are their respective data sizes. Before calling, ensure the context is initialized and the data pointers and sizes are valid. An error code is returned on failure. After successful loading, model initialization and inference can be performed.

API	**rknn3_load_model_from_data**
Function	Loads an RKNN3 model from memory data.
Parameters	rknn3_context context: The RKNN3 context handle.
	const void* model_data: Pointer to the model data in memory.
	uint64_t model_size: The size of the model data in bytes.
	const void* weight_data: Pointer to the weight data in memory.
	uint64_t weight_size: The size of the weight data in bytes.
Return	int Status code: see 5.2. Status Codes.

5.1.7. rknn3_model_init

This interface initializes the runtime configuration for a loaded model, including priority, timeout, and core mask. config is a pointer to the configuration parameters. Before calling, ensure the model has been loaded and the config parameters are valid. An error code is returned on failure. After successful initialization, the model is ready for inference.

API	**rknn3_model_init**
Function	Initializes an RKNN3 model.
Parameters	rknn3_context context: The RKNN3 context handle.
	rknn3_config* config: Model configuration parameters.
Return	int Status code: see 5.2. Status Codes.

5.1.8. rknn3_dup_context

This interface duplicates an existing RKNN3 context, facilitating context reuse in multi-threaded or multi-instance scenarios. The context_in is the source context, and context_out is a pointer to the destination context. Before calling, ensure the source context is valid and the destination pointer is not null. After duplication, each context's resources must be managed and released separately.

API	**rknn3_dup_context**
Function	Duplicates an existing RKNN3 context.

API	rknn3_dup_context
Parameters	rknn3_context context_in: The source RKNN3 context to be duplicated.
	rknn3_context* context_out: Pointer to receive the duplicated RKNN3 context.
Return	int Status code: see 5.2. Status Codes.

5.1.9. rknn3_destroy

This interface destroys an RKNN3 runtime context and releases associated resources. After calling, the context handle becomes invalid and cannot be used for other operations. Destroying the same context multiple times will lead to undefined behavior; ensure each context is destroyed only once.

API	**rknn3_destroy**
Function	Destroys the RKNN3 runtime context and releases resources.
Parameters	rknn3_context context: The RKNN3 context handle to be destroyed.
Return	int Status code: see 5.2. Status Codes.

5.1.10. rknn3_query

This interface queries various information about the model and runtime, such as input/output attributes, and SDK version. cmd specifies the query type; see 5.4.1. rknn3_query_cmd. info is a pointer to the result buffer, and size is the buffer size. Before calling, ensure context and info are valid and size matches the size of the structure corresponding to the query type.

API	**rknn3_query**
Function	Queries information or status from RKNN3.
Parameters	rknn3_context context: The context of the RKNN3 model.
	rknn3_query_cmd cmd: The query command type.
	void* info: Pointer to a buffer to store the query result.
	uint64_t size: The size of the info buffer in bytes.
Return	int Status code: see 5.2. Status Codes.
Note	This function is used to query various information about the RKNN3 model and runtime, such as SDK version, device info, model info, etc. The specific information returned depends on the specified query command.

5.1.11. rknn3_run

This interface performs synchronous inference. inputs and outputs are arrays of input and output tensors, and n_inputs and n_outputs are their respective counts. Before calling, ensure context, inputs, and outputs are correctly allocated and configured. The call will block until inference is complete and returns a status code. Memory for input and output tensors must be pre-allocated by the user.

API	**rknn3_run**
Function	Performs RKNN3 model inference.
Parameters	rknn3_context context: The RKNN3 context handle from rknn3_init.
	const rknn3_tensor inputs[]: Input tensor array containing input data.
	uint32_t n_inputs: The number of input tensors.
	rknn3_tensor outputs[]: Output tensor array to store inference results.
	uint32_t n_outputs: The number of output tensors.
Return	int Status code: see 5.2. Status Codes.
Note	This function performs inference using the specified RKNN3 model. It takes input data via the inputs array and writes results to the outputs array. The input and output tensors must be correctly allocated and configured before calling this function.

5.1.12. rknn3_run_async

This interface performs asynchronous inference, with parameters identical to rknn3_run. The call returns immediately, and inference proceeds in the background. It must be used in conjunction with rknn3_wait to wait for completion. Asynchronous inference is suitable for scenarios requiring concurrency or pipelining.

API	**rknn3_run_async**
Function	Performs asynchronous RKNN3 model inference.
Parameters	rknn3_context context: The RKNN3 context handle from rknn3_init.
	const rknn3_tensor inputs[]: Input tensor array containing input data.
	uint32_t n_inputs: The number of input tensors.
	rknn3_tensor outputs[]: Output tensor array to store inference results.
	uint32_t n_outputs: The number of output tensors.
Return	int Status code: see 5.2. Status Codes.
Note	This function performs asynchronous inference using the specified RKNN3 model. It takes input data via the inputs array and writes results to the outputs array. The input and output tensors must be correctly allocated and configured before calling this function.

5.1.13. rknn3_wait

This interface blocks and waits for an asynchronous inference to complete. context is the context handle of the model instance. It only needs to be called after rknn3_run_async. This interface is not needed for synchronous inference.

API	**rknn3_wait**
Function	Waits for inference/execution to complete.
Parameters	rknn3_context context: The context handle of the RKNN3 model instance.
Return	int Status code: see 5.2. Status Codes.
Note	This function blocks until the inference or execution on the RKNN3 device is finished.

5.1.14. rknn3_create_mem_from_phys

Creates a tensor memory object from a physical address. Before calling, ensure the context is valid and related parameters (like address, size) are correct. The allocated memory must be released via rknn3_destroy_mem when no longer in use to prevent memory leaks.

API	**rknn3_create_mem_from_phys**
Function	Creates a tensor memory object from a physical address.
Parameters	rknn3_context context: The RKNN3 context handle.
	uint64_t phys_addr: The physical address of the memory.
	void* virt_addr: The virtual address of the memory.
	uint64_t size: The size of the memory in bytes.
Return	rknn3_tensor_mem* A pointer to the created tensor memory object, or NULL if creation fails.
Note	This function creates a tensor memory object from the provided physical and virtual addresses. The memory must be pre-allocated, and the physical/virtual addresses must be valid.

5.1.15. rknn3_create_mem_from_fd

Creates a tensor memory object from a file descriptor. Before calling, ensure the context is valid and related parameters (like fd, size) are correct. The allocated memory must be released via rknn3_destroy_mem when no longer in use to prevent memory leaks.

API	**rknn3_create_mem_from_fd**
Function	Creates a tensor memory object from a file descriptor.
Parameters	rknn3_context context: The RKNN3 context handle.
	int32_t fd: The file descriptor of the memory.
	void* virt_addr: The virtual address of the memory.
	uint64_t size: The size of the memory in bytes.
	uint64_t offset: The offset from the beginning of the memory referenced by fd.
Return	rknn3_tensor_mem* A pointer to the created tensor memory object, or NULL if creation fails.

5.1.16. rknn3_create_mem

This interface is used to allocate or register memory for a tensor. Before calling, ensure the context is valid. The core_id for input and output tensors should be set to the core_id member of the rknn3_tensor_attr struct obtained from the rknn3_query interface. Allocated memory must be released via rknn3_destroy_mem when no longer needed to avoid memory leaks.

API	**rknn3_create_mem**
Function	Creates a memory buffer for an RKNN3 tensor.
Parameters	rknn3_context context: The RKNN3 context handle.
	uint64_t size: The size of the memory to allocate in bytes.
	int32_t core_id: The target NPU core ID for memory allocation.
	rknn3_mem_alloc_flags flags: Memory allocation flags to control allocation behavior.
Return	rknn3_tensor_mem* A pointer to the allocated tensor memory, or NULL if allocation fails.
Note	This function allocates memory that can be used for RKNN3 tensor operations. The memory is allocated on the specified core with the given flags.

5.1.17. rknn3_destroy_mem

This interface releases tensor memory allocated or registered via the RKNN3 API. Before calling, ensure that context and mem are valid. Releasing the same memory multiple times will result in undefined behavior.

API	**rknn3_destroy_mem**
Function	Destroys memory allocated for an RKNN3 tensor.
Parameters	rknn3_context context: The RKNN3 context handle.
	rknn3_tensor_mem* mem: Pointer to the tensor memory structure to be destroyed.
Return	int Status code: see 5.2. Status Codes.

5.1.18. rknn3_mem_sync

This interface synchronizes memory data between the CPU and the device. The mode specifies the synchronization direction and is often used before and after inference to ensure data consistency. Before calling, ensure context and mem are valid and mode is set correctly. Failure to synchronize may lead to abnormal inference results.

API	**rknn3_mem_sync**
Function	Synchronizes memory data between the CPU and the device.
Parameters	rknn3_context context: The context of the RKNN3 model.
	rknn3_tensor_mem* mem: The memory handle of the tensor.
	rknn3_mem_sync_mode mode: Mode indicating the flushing of CPU cache and DDR data. **RKNN3_MEMORY_SYNC_TO_DEVICE**: Flushes CPU cache data to DDR. Typically used after the CPU writes to memory and before the NPU accesses the same memory. **RKNN3_MEMORY_SYNC_FROM_DEVICE**: Synchronizes DDR data to the CPU cache. Typically used after the NPU writes to memory to invalidate the cache, forcing the CPU to re-read data from DDR on its next access. **RKNN3_MEMORY_SYNC_BIDIRECTIONAL**: Flushes CPU cache data to DDR and simultaneously forces the CPU to re-read data from DDR.
Return	int Status code: see 5.2. Status Codes.

5.1.19. rknn3_set_shape

This interface sets the current shape for a model with dynamic inputs. shape_id is a predefined shape ID within the model. Before calling, ensure the context is valid and the shape_id is legal. This function only needs to be called for dynamic input models; it is not necessary for static models.

API	**rknn3_set_shape**
Function	Sets the model shape for dynamic inputs.
Parameters	rknn3_context context: The context handle of the RKNN3 model.
	int32_t shape_id: The ID of the shape to set.
Return	int Status code: see 5.2. Status Codes.

5.1.20. rknn3_set_kvcache_mem

This interface sets the KV cache memory for specified NPU cores. mem is an array of KV cache memory objects, npu_core_indices is an array of core indices, and n_core is the number of cores. Before calling, ensure that context, mem, and npu_core_indices are valid and n_core is greater than 0. mem and npu_core_indices must correspond one-to-one.

API	**rknn3_set_kvcache_mem**
Function	Sets the KV cache memory.
Parameters	rknn3_context context: The context handle of the RKNN3 model.
	rknn3_tensor_mem* mem[]: KV cache memory information.
	int* npu_core_indices: NPU core indices.
	int n_core: The number of NPU cores for which to set the KV cache.
Return	int Status code: see 5.2. Status Codes.

5.1.21. rknn3_set_internal_mem

Sets user-allocated internal Tensor memory for multiple cores. The core_id field of each mem object specifies the target core.

API	rknn3_set_internal_mem
Function	Sets user-allocated internal Tensor memory for multiple cores.
Parameters	rknn3_context context: The RKNN3 context.
	rknn3_tensor_mem* mem[]: Array of user-allocated memory objects.
	uint32_t n_core: The number of cores.
Return	int Status code: see 5.2. Status Codes.

5.1.22. rknn3_dump_features

Runs the model layer by layer and dumps each intermediate Tensor to the specified directory as `.npy` files. If no output tensors are provided, they are allocated internally.

API	rknn3_dump_features
Function	Executes the model layer by layer and saves intermediate tensors.
Parameters	rknn3_context context: RKNN3 context handle.
	const rknn3_tensor inputs[]: Array of input tensors.
	uint32_t n_inputs: Number of input tensors.
	rknn3_tensor outputs[]: Optional output tensor array; set to NULL or provide zero count to allocate internally.
	uint32_t n_outputs: Number of output tensors; use 0 to enable internal allocation.
	const char* dump_dir: Directory path for saving `.npy` files.
Return	int Status code: see 5.2. Status Codes.
Note	Ensure `dump_dir` is writable; active cores used during dumping come from `run_core_mask` set by rknn3_model_init.

5.1.23. rknn3_find_devices

This interface retrieves a list of available RKNN3 devices in the current system. pdevs is a pointer to the device information structure. Before calling, ensure pdevs is valid. A return value of 0 indicates no devices or that the function is not implemented.

API	rknn3_find_devices
Function	Gets a list of available RKNN3 devices.
Parameters	rknn3_devices* pdevs: A pointer to a structure that will receive the device list.
Return	int Status code: see 5.2. Status Codes.
Note	This function populates the provided rknn3_devices structure with information about all available RKNN3 devices on the system.

5.1.24. rknn3_session_init

This interface initializes a new RKNN3 session, supporting advanced features like LLMs. context is the context, params are the session parameters, and n_params is the number of parameters. Before calling, ensure context and params are valid. It returns a session handle on success, or NULL on failure. The session must be released via rknn3_session_destroy.

API	**rknn3_session_init**
Function	Initializes a new RKNN3 session with specified parameters.
Parameters	rknn3_context context: The RKNN3 context pointer for the session.
	rknn3_llm_param* params: Pointer to an rknn3_llm_param structure containing session configuration parameters.
	int n_params: The number of parameters.
Return	rknn3_session* Returns a session handle on success, or NULL if initialization fails.

5.1.25. rknn3_session_enable_lora

Enables LoRA for an RKNN3 session. Before calling, ensure session and lora are valid.

API	**rknn3_session_enable_lora**
Function	Enables LoRA for an RKNN3 session.
Parameters	rknn3_session* session: The RKNN3 session pointer.
	rknn3_lora* lora: The LoRA pointer to enable.
Return	int Status code: see 5.2. Status Codes.

5.1.26. rknn3_session_disable_lora

Disables LoRA for an RKNN3 session. Before calling, ensure session and lora are valid.

API	**rknn3_session_disable_lora**
Function	Disables LoRA for an RKNN3 session.
Parameters	rknn3_session* session: The RKNN3 session pointer.
	rknn3_lora* lora: The LoRA pointer to disable.
Return	int Status code: see 5.2. Status Codes.

5.1.27. rknn3_session_query_lora

Queries LoRA information from an RKNN3 session.

API	rknn3_session_query_lora
Function	Queries LoRA information from an RKNN3 session.
Parameters	rknn3_session* session: The RKNN3 session pointer.
	rknn3_lora** lora: A pointer to a pointer to store the LoRA information array.
	int* n_lora: A pointer to store the number of LoRA entries.
Return	int Status code: see 5.2. Status Codes.

5.1.28. rknn3_session_set_kvcache_policy

Sets the KV cache policy for an RKNN3 session. This is applicable for models like Transformers that require a caching mechanism. Before calling, ensure the session is valid and related parameters are correct. KV cache operations must be coordinated with the inference flow to avoid cache inconsistencies.

API	rknn3_session_set_kvcache_policy
Function	Sets the KV cache policy for an RKNN3 session.
Parameters	rknn3_session* session: The RKNN3 session handle.
	rknn3_kvcache_policy policy: The KV cache policy to set.
	rknn3_kvcache_policy_param* param: Parameters specifying the KV cache policy.
Return	int Status code: see 5.2. Status Codes.

5.1.29. rknn3_session_clear_kvcache

Clears the KV cache of an RKNN3 session according to a specified policy.

API	rknn3_session_clear_kvcache
Function	Clears the KV cache of an RKNN3 session according to a specified policy.
Parameters	rknn3_session* session: The RKNN3 session handle pointer.
	rknn3_kvcache_clear_policy clear: The policy for clearing the KV cache, defining how the cache is cleared.
Return	int Status code: see 5.2. Status Codes.

5.1.30. rknn3_session_load_kvcache

Loads the KV cache from a specified path.

API	rknn3_session_load_kvcache
Function	Loads the KV cache from a specified path.
Parameters	rknn3_session* session: The RKNN3 session pointer.
	const char* kvcache_path: The path to the KV cache file.
	int64_t size: The length of the KV cache file path, i.e., strlen(kvcache_path).
Return	int Status code: see 5.2. Status Codes.

5.1.31. rknn3_session_save_kvcache

Saves the KV cache to a specified path.

API	rknn3_session_save_kvcache
Function	Saves the KV cache to a specified path.
Parameters	rknn3_session* session: The RKNN3 session pointer.
	char* kvcache_path: The path to save the KV cache.
Return	int Status code: see 5.2. Status Codes.

5.1.32. rknn3_session_query_state

This interface queries the current running state of a session, including the number of generated tokens, KV policy, etc. session is the session pointer, and state is a pointer to the state structure. Before calling, ensure session and state are valid. It can be used to monitor inference progress and session status.

API	rknn3_session_query_state
Function	Queries the current state of an RKNN3 session.
Parameters	rknn3_session* session: The RKNN3 session pointer to query.
	RKLLMRunState* state: A pointer to store the queried run state.
Return	int Status code: see 5.2. Status Codes.

5.1.33. rknn3_session_set_chat_template

This interface sets the chat template for an LLM, including the system prompt, prefix, and postfix. session is the session handle, and system_prompt, prompt_prefix, prompt_postfix are the template contents. Before calling, ensure the session is valid and the string contents meet the model's requirements. Custom templates help in adjusting the model's behavior and conversational style.

API	**rknn3_session_set_chat_template**
Function	Sets the chat template for an LLM, including system prompt, prefix, and postfix.
Parameters	rknn3_session* session: The RKNN3 session handle.
	const char* system_prompt: A system prompt that defines the context or behavior of the language model.
	const char* prompt_prefix: A prefix to be added before user input in a chat.
	const char* prompt_postfix: A suffix to be added after user input in a chat.
Return	int Status code: see 5.2. Status Codes.

5.1.34. rknn3_session_set_callback

This interface sets callback functions for a session, supporting custom result handling, sampling, tokenization, embeddings, etc. session is the session pointer, and callback is a pointer to the callback structure. Before calling, ensure session and callback are valid. The callback functions are triggered during the inference process, allowing users to customize the inference flow.

API	**rknn3_session_set_callback**
Function	Sets callback functions for an RKNN3 session.
Parameters	rknn3_session* session: The RKNN3 session instance pointer.
	RKLLMCallback* callback: Pointer to an RKLLMCallback structure containing the callback functions.
Return	int Status code: see 5.2. Status Codes.

5.1.35. rknn3_session_run

Runs inference using the provided inputs and parameters. Before calling, ensure session, inputs, and param are valid.

API	rknn3_session_run
Function	Runs inference using the provided inputs and parameters.
Parameters	rknn3_session* session: The RKNN3 session handle pointer.
	rknn3_llm_input inputs[]: Input tensor array containing input data.
	uint32_t n_inputs: The number of provided input tensors.
	rknn3_llm_infer_param* param: Pointer to inference parameter configuration.
Return	int Status code: see 5.2. Status Codes.

5.1.36. rknn3_session_run_async

Asynchronously runs inference for a large language model session. Before calling, ensure session, inputs, and param are valid. Asynchronous inference requires using callbacks or a waiting mechanism to get results.

API	rknn3_session_run_async
Function	Asynchronously runs inference for a large language model session.
Parameters	rknn3_session* session: The RKNN3 session handle pointer.
	rknn3_llm_input inputs[]: The model's input tensor array.
	uint32_t n_inputs: The number of input tensors.
	rknn3_llm_infer_param* param: Pointer to inference parameter configuration.
Return	int Status code: see 5.2. Status Codes.
Note	This function performs asynchronous inference for a given LLM session. It allows for non-blocking execution of the model with the provided inputs and parameters.

5.1.37. rknn3_session_stop

Terminates the current session inference. After calling, only the current inference is terminated; new inferences can still be initiated later. This is suitable for scenarios where long-running inferences need to be interrupted.

API	rknn3_session_stop
Function	Terminates the current session inference.
Parameters	rknn3_session* session: The RKNN3 session pointer.

API	rknn3_session_stop
Return	int Status code: see 5.2. Status Codes.

5.1.38. rknn3_session_destroy

This interface destroys an RKNN3 session and releases associated resources. session is the session pointer. After calling, the session pointer becomes invalid and should not be used for other operations. Ensure each session is destroyed only once to avoid resource leaks or double-freeing.

API	rknn3_session_destroy
Function	Destroys an RKNN3 session and releases associated resources.
Parameters	rknn3_session* session: The RKNN3 session pointer to be destroyed.
Return	int Status code: see 5.2. Status Codes.
Note	After calling this function, the session pointer becomes invalid and should not be used again.

5.1.39. rknn3_session_set_function_tools

This interface sets function tools descriptions for a session to help the model call tools during conversation. Ensure session and tools are valid before calling.

API	rknn3_session_set_function_tools
Function	Sets function tools descriptions for an LLM session.
Parameters	rknn3_session* session: The RKNN3 session handle.
	const char* tools: Function tools description string.
Return	int Status code: see 5.2. Status Codes.

5.1.40. rknn3_profile_ops

Runs the model layer by layer and prints operator profile information, including time, cycles, and bandwidth. Output tensors are allocated internally when not provided.

API	rknn3_profile_ops
Function	Runs the model layer by layer and prints operator profiling information.
Parameters	rknn3_context context: The RKNN3 context handle.
	const rknn3_tensor inputs[]: Input tensor array.
	uint32_t n_inputs: Number of input tensors.
	rknn3_tensor outputs[]: Output tensor array, can be NULL.

API	rknn3_profile_ops
	uint32_t n_outputs: Number of output tensors, can be 0 to enable internal output allocation.
	uint32_t log_level: Log level (0: summary only; 1: per-op details + summary; 2: per-op details with time/cycles/bandwidth + summary).
Return	int Status code: see 5.2. Status Codes.
Note	Active cores used during profiling come from `run_core_mask` set by rknn3_model_init.

5.1.41. rknn3_profile_mem

Prints memory usage information for each NPU core. It should be called after rknn3_model_init.

API	rknn3_profile_mem
Function	Prints memory usage information for each NPU core.
Parameters	rknn3_context context: The RKNN3 context handle.
Return	int Status code: see 5.2. Status Codes.
Note	It is recommended to call after rknn3_model_init to ensure memory is allocated.

5.1.42. rknn3_register_custom_ops_plugins

Registers custom operator plugins (currently supports postprocess plugins). This interface loads a plugin and registers it to the specified context.

API	rknn3_register_custom_ops_plugins
Function	Registers custom operator plugins.
Parameters	rknn3_context ctx: The RKNN3 context handle.
	const char* plugin_path: Path to the plugin shared library.
	int64_t size: Reserved field, can be set to 0.
Return	int Status code: see 5.2. Status Codes.

5.1.43. rknn3_register_custom_op_func

Custom operator registration function pointer type for registering custom ops inside a plugin.

Function Pointer Type	rknn3_register_custom_op_func
Function	Returns a custom operator descriptor pointer based on the op index.

Function Pointer Type	rknn3_register_custom_op_func
Parameters	int op_index: Operator index (starting from 0).
Return	rknn3_custom_op*: Pointer to the custom op descriptor; returns NULL if the index is not available.

5.1.44. LLMResultCallback

LLM result callback function type.

Function Pointer Type	LLMResultCallback
Function	Callback function to handle LLM generation results.
Parameters	void* userdata: User-defined data pointer. RKLLMResult* result: Pointer to the LLM result. LLMCallState state: LLM call state (e.g., complete, error).
Return	int: Returns 0 on success, non-zero on error.

5.1.45. LLMSamplingCallback

LLM sampling callback function type.

Function Pointer Type	LLMSamplingCallback
Function	Callback function to handle sampling from logits.
Parameters	void* userdata: User-defined data pointer. float16* logits: Pointer to the logits array. char* logits_name: The name of the logits.
Return	int: Returns the selected token ID (>=0) on success, or a negative error code on failure. See 5.2. Status Codes.

5.1.46. LLMGetEmbedCallback

Callback function type for getting LLM embedding vectors.

Function Pointer Type	LLMGetEmbedCallback
Function	Callback function to get LLM embedding vectors.
Parameters	void* userdata: User-defined data pointer. int32_t* tokens: Array of token IDs. uint64_t num_tokens: Number of tokens in the tokens array. void* embed: Pointer to the buffer to store the embedding output. uint64_t len: Length of the embedding buffer in bytes.
Return	int Status code: see 5.2. Status Codes.

5.1.47. LLMTokenizerCallback

LLM tokenizer callback function type.

Function Pointer Type	LLMTokenizerCallback
Function	Callback function to handle text tokenization.
Parameters	void* userdata: User-defined data pointer. const char* text: Pointer to the input text string. int32_t text_len: Length of the text. int32_t* tokens: Pointer to the token ID array. int32_t n_tokens_max: Maximum number of tokens to generate.
Return	int: Returns the number of generated tokens (>=0) on success, or a negative error code on failure. See 5.2. Status Codes.

5.1.48. LLMOutputCallback

Callback function type for retrieving output tensors.

Function Pointer Type	LLMOutputCallback
Function	Callback function to retrieve output tensors.
Parameters	void* userdata: User-defined data pointer. rknn3_tensor* output_tensors: Pointer to the output tensor array. uint32_t n_output_tensors: Number of output tensors. LLMOutputCallbackState state: Output callback state.
Return	int: Returns 0 on success, non-zero on error.

5.1.49. rknn3_im_mem_create

This interface creates an RKNN3 image memory object `rknn3_im_mem`. It is typically used for image preprocessing and postprocessing before or after model inference.

API	rknn3_im_mem_create
Function	Allocates an RKNN3 image memory object on the device.
Parameters	rknn3_context context: RKNN3 context handle.
	int width: Image width in pixels.
	int height: Image height in pixels.
	rknn3_im_fmt format: Image format, see 5.4.15. rknn3_im_fmt.
	int size: Image memory size in bytes. If 0, it is calculated based on width, height, and format.
	int core_id: Target NPU core ID for memory allocation.

API	rknn3_im_mem_create
	rknn3_mem_alloc_flags flags: Memory allocation flags.
	rknn3_im_mem* im_mem: Pointer to the created `rknn3_im_mem` object.
Return	int Status code: see 5.2. Status Codes.
Note	On success, `im_mem` is initialized and must later be released by `rknn3_im_mem_destroy`.

5.1.50. rknn3_im_mem_destroy

This interface destroys an RKNN3 image memory object created by `rknn3_im_mem_create` and releases the associated device memory.

API	rknn3_im_mem_destroy
Function	Destroys an RKNN3 image memory object and releases device memory.
Parameters	rknn3_context context: RKNN3 context handle.
	rknn3_im_mem* im_mem: Pointer to the `rknn3_im_mem` object to destroy.
Return	int Status code: see 5.2. Status Codes.
Note	After this call, `im_mem` must not be used again.

5.1.51. rknn3_im_cvt_color

This interface converts the color format of an image stored in RKNN3 image memory. It can be used to convert between different RGB/YUV formats before feeding data into the model.

API	rknn3_im_cvt_color
Function	Converts the color format of an image in RKNN3 image memory.
Parameters	rknn3_context context: RKNN3 context handle.
	rknn3_im_mem* src: Source image memory object.
	rknn3_im_mem* dst: Destination image memory object.
Return	int Status code: see 5.2. Status Codes.
Note	`src` and `dst` must have been created by `rknn3_im_mem_create`.

5.2. Status Codes

Status Code	Value	Description
RKNN3_SUCCESS	0	Execution successful.
RKNN3_ERR_FAIL	-1	Execution failed.
RKNN3_ERR_ARGUMENT_INVALID	-2	Invalid argument.
RKNN3_ERR_MODEL_INVALID	-3	Invalid model.
RKNN3_ERR_CTX_INVALID	-4	Invalid context.
RKNN3_ERR_RUN_TASK_FAILED	-5	Task run failed.
RKNN3_ERR_OUT_OF_MEMORY	-6	Out of memory.
RKNN3_ERR_TIMEOUT	-7	Execution timed out.
RKNN3_ERR_INPUT_INVALID	-8	Invalid input.
RKNN3_ERR_OUTPUT_INVALID	-9	Invalid output.
RKNN3_ERR_DEVICE_UNAVAILABLE	-10	Device unavailable.
RKNN3_ERR_DEVICE_UNMATCH	-11	Device mismatch.
RKNN3_ERR_TARGET_PLATFORM_UNMATCH	-12	Target platform mismatch.
RKNN3_ERR_COMMUNICATION	-13	Communication error.
RKNN3_ERR_MEM_SYNC_FAILED	-14	Memory synchronization failed.
RKNN3_WARN_NPU_CORE_UNUSED	-100	NPU core is unused. This is only a warning and does not affect model execution.

5.3. Constants

5.3.1. Tensor-Related Constants

Constant Name	Value	Description
RKNN3_MAX_DIMS	16	Maximum number of dimensions for a tensor.
RKNN3_MAX_STRIDE_DIMS	17	Maximum number of stride dimensions for a tensor.
RKNN3_MAX_NAME_LEN	256	Maximum length of a tensor name.
RKNN3_MAX_DYNAMIC_SHAPE_NUM	512	Maximum number of dynamic shapes per input.
RKNN3_MAX_LORA_NUM	128	Maximum number of LoRA instances.
RKNN3_MAX_SPECIAL_BOS_ID_NUM	64	maximum number of special Begin-Of-Sequence (BOS) token.
RKNN3_MAX_SPECIAL_EOS_ID_NUM	64	maximum number of special End-Of-Sequence (EOS) token.

5.3.2. Device-Related Constants

Constant Name	Value	Description
RKNN3_MAX_DEVS	64	Maximum number of devices.
RKNN3_MAX_DEV_LEN	64	Maximum length of a device ID/type.
RKNN3_MAX_NPU_NODE_NUM	128	Maximum number of NPU nodes.

5.4. Enumerations

5.4.1. rknn3_query_cmd

Query command enumeration, used in the rknn3_query function.

Enum Value	Description
RKNN3_QUERY_IN_OUT_NUM	Query the number of input and output tensors.
RKNN3_QUERY_INPUT_ATTR	Query the attributes of an input tensor.
RKNN3_QUERY_OUTPUT_ATTR	Query the attributes of an output tensor.
RKNN3_QUERY_SDK_VERSION	Query the SDK and driver versions.
RKNN3_QUERY_CORE_MEM_SIZE	Query the weight and internal memory size for each core.
RKNN3_QUERY_NATIVE_INPUT_ATTR	Query the attributes of a native input tensor.
RKNN3_QUERY_NATIVE_OUTPUT_ATTR	Query the attributes of a native output tensor.
RKNN3_QUERY_DEVICE_MEM_INFO	Query the attributes of RKNN3 memory information.
RKNN3_QUERY_CORE_NUMBER	Query the number of cores.
RKNN3_QUERY_ALLOCATION_INFO	Query allocation information.
RKNN3_QUERY_DYNAMIC_SHAPE_CONFIG	Query the complete dynamic shape configuration.
RKNN3_QUERY_DYNAMIC_SHAPE_INFO	Query all supported shape combinations.
RKNN3_QUERY_LLM_CONFIG	Query LLM-specific configurations, such as the chat template.
RKNN3_QUERY_POSTPROCESS_IN_OUT_NUM	Query the number of postprocess inputs and outputs, valid only when postprocess is enabled.
RKNN3_QUERY_POSTPROCESS_OUTPUT_ATTR	Query postprocess output tensor attributes, valid only when postprocess is enabled.
RKNN3_QUERY_POSTPROCESS_DYNAMIC_SHAPE_INFO	Query postprocess dynamic shape information, valid only when postprocess is enabled.
RKNN3_QUERY_CMD_MAX	The maximum value of the query command enumeration.

5.4.2. rknn3_tensor_type

Tensor data type enumeration.

Enum Value	Description
RKNN3_TENSOR_FLOAT32	float32 type
RKNN3_TENSOR_FLOAT16	float16 type
RKNN3_TENSOR_INT8	int8 type
RKNN3_TENSOR_UINT8	uint8 type
RKNN3_TENSOR_INT16	int16 type
RKNN3_TENSOR_UINT16	uint16 type
RKNN3_TENSOR_INT32	int32 type
RKNN3_TENSOR_UINT32	uint32 type
RKNN3_TENSOR_INT64	int64 type
RKNN3_TENSOR_UINT64	uint64 type
RKNN3_TENSOR_BOOL	bool type
RKNN3_TENSOR_INT4	int4 type
RKNN3_TENSOR_TYPE_MAX	The maximum value of the tensor data type enumeration.

5.4.3. rknn3_tensor_qnt_type

Quantization type enumeration.

Enum Value	Description
RKNN3_TENSOR_QNT_NONE	No quantization.
RKNN3_TENSOR_PER_LAYER_SYMMETRIC	Per-layer symmetric quantization.
RKNN3_TENSOR_PER_LAYER_ASYMMETRIC	Per-layer asymmetric quantization.
RKNN3_TENSOR_PER_CHANNEL_SYMMETRIC	Per-channel symmetric quantization.
RKNN3_TENSOR_PER_CHANNEL_ASYMMETRIC	Per-channel asymmetric quantization.
RKNN3_TENSOR_PER_GROUP_SYMMETRIC	Per-group symmetric quantization.
RKNN3_TENSOR_PER_GROUP_ASYMMETRIC	Per-group asymmetric quantization.
RKNN3_TENSOR_QNT_MAX	The maximum value of the quantization type enumeration.

5.4.4. rknn3_tensor_layout

Tensor data layout enumeration.

Enum Value	Description
RKNN3_TENSOR_UNDEFINED	Undefined.
RKNN3_TENSOR_NCHW	Data layout is NCHW.
RKNN3_TENSOR_NHWC	Data layout is NHWC.
RKNN3_TENSOR_NC1HWC2	Data layout is NC1HWC2.
RKNN3_TENSOR_CHWN	Reserved value, not currently in use.
RKNN3_TENSOR_HWIO	Reserved value, not currently in use.
RKNN3_TENSOR_OIHW	Reserved value, not currently in use.
RKNN3_TENSOR_O1I1HWI2O2	Reserved value, not currently in use.
RKNN3_TENSOR_LAYOUT_MAX	The maximum value of the tensor data layout enumeration.

5.4.5. rknn3_mem_alloc_flags

Memory allocation flags for creating RKNN3 tensor memory.

Enum Value	Description
RKNN3_FLAG_MEMORY_FLAGS_DEFAULT	Same as RKNN3_FLAG_MEMORY_CACHEABLE.
RKNN3_FLAG_MEMORY_CACHEABLE	Create cacheable memory.
RKNN3_FLAG_MEMORY_NON_CACHEABLE	Create non-cacheable memory.

5.4.6. rknn3_mem_sync_mode

Memory synchronization mode for the rknn3_mem_sync function.

Enum Value	Description
RKNN3_MEMORY_SYNC_TO_DEVICE	Flushes CPU cache data to DDR. Typically used after the CPU writes to memory and before the NPU accesses the same memory to write cached data back to DDR.
RKNN3_MEMORY_SYNC_FROM_DEVICE	Synchronizes DDR data to the CPU cache. Typically used after the NPU writes to memory to invalidate the cache, forcing the CPU to re-read data from DDR on its next access.
RKNN3_MEMORY_SYNC_BIDIRECTIONAL	Flushes CPU cache data to DDR and simultaneously forces the CPU to re-read from DDR.

5.4.7. rknn3_core_mask

Mode for running on the target NPU core(s).

Enum Value	Description
RKNN3_NPU_CORE_AUTO	Default. Automatically schedules the model to run on a currently idle NPU core.
RKNN3_NPU_CORE_0	Run on NPU core 0.
RKNN3_NPU_CORE_1	Run on NPU core 1.
RKNN3_NPU_CORE_2	Run on NPU core 2.
RKNN3_NPU_CORE_3	Run on NPU core 3.
RKNN3_NPU_CORE_4	Run on NPU core 4.
RKNN3_NPU_CORE_5	Run on NPU core 5.
RKNN3_NPU_CORE_6	Run on NPU core 6.
RKNN3_NPU_CORE_7	Run on NPU core 7.
RKNN3_NPU_CORE_ALL	Run on all NPU cores.

5.4.8. rknn3_kvcache_policy

KV cache policy.

Enum Value	Description
RKNN3_KVCACHE_POLICY_DEFAULT	Default policy, equivalent to RKNN3_KVCACHE_POLICY_RECURRENT.
RKNN3_KVCACHE_POLICY_RECURRENT	Use a recurrent caching policy.
RKNN3_KVCACHE_POLICY_NORMAL	Use a normal caching policy, only using the KV cache within max_context_len.

5.4.9. rknn3_kvcache_clear_policy

Policy for clearing the KV cache.

Enum Value	Description
RKNN3_KVCACHE_CLEAR_ALL	Completely clear all KV cache entries.
RKNN3_KVCACHE_KEEP_SYSTEM_PROMPT	Clear KV cache but keep the system prompt.

5.4.10. rknn3_llm_input_type

Defines the types of input that can be provided to the LLM.

Enum Value	Description
RKNN3_LLM_INPUT_PROMPT	The input is a text prompt.
RKNN3_LLM_INPUT_TOKEN	The input is a series of tokens.
RKNN3_LLM_INPUT_EMBED	The input is an embedding vector.
RKNN3_LLM_INPUT_MULTIMODAL	Multimodal input.
RKNN3_LLM_INPUT_AUX	Other input types.
RKNN3_LLM_INPUT_MAX	The maximum value of the input type enumeration.

5.4.11. LLMCallState

Describes the possible states of an LLM call.

Enum Value	Description
RKLLM_RUN_NORMAL	The LLM call is in a normal running state.
RKLLM_RUN_WAITING	The LLM call is waiting for a complete UTF-8 character.
RKLLM_RUN_FINISH	The LLM call has finished execution.
RKLLM_RUN_STOP	The LLM call was stopped by the user.
RKLLM_RUN_MAX_NEW_TOKEN_REACHED	The maximum number of new tokens has been reached.
RKLLM_RUN_ERROR	An error occurred during the LLM call.

5.4.12. rknn3_mem_type

Memory type enumeration.

Enum Value	Description
RKNN3_MEMORY_TYPE_NPU_DRAM	NPU DRAM memory.
RKNN3_MEMORY_TYPE_EXT_DDR	External DDR memory.

5.4.13. rknn3_kvcache_dtype

KV cache data type. These are internally defined data types that affect KV cache size and management; users do not need to parse them.

Enum Value	Description
RKNN3_KVCACHE_DTYPE_UNDEFINED	Undefined.
RKNN3_KVCACHE_DTYPE_INT4_TO_F16	INT4 to FLOAT16.
RKNN3_KVCACHE_DTYPE_INT4_TO_F8	INT4 to FLOAT8.
RKNN3_KVCACHE_DTYPE_INT8_TO_F16	INT8 to FLOAT16.
RKNN3_KVCACHE_DTYPE_FLOAT4_TO_F16	FLOAT4 to FLOAT16.
RKNN3_KVCACHE_DTYPE_FLOAT4_TO_F8	FLOAT4 to FLOAT8.
RKNN3_KVCACHE_DTYPE_FLOAT8_TO_F16	FLOAT8 to FLOAT16.
RKNN3_KVCACHE_DTYPE_FLOAT8_TO_F8	FLOAT8 to FLOAT8.
RKNN3_KVCACHE_DTYPE_FLOAT16	FLOAT16.

5.4.14. rknn3_kvcache_store_method

KV cache storage method. These are internally defined data types that affect KV cache size and management; users do not need to parse them.

Enum Value	Description
RKNN3_KVCACHE_STORE_METHOD_UNDEFINED	Undefined.
RKNN3_KVCACHE_STORE_METHOD_NORMAL	Normal storage.
RKNN3_KVCACHE_STORE_METHOD_GROUP_QUANT	Group quantized storage.

5.4.15. rknn3_im_fmt

Image format enumeration used by RKNN3 image-related interfaces.

Enum Value	Description
RKNN3_IM_FMT_RGB888	RGB888 format, commonly used in general image preprocessing.
RKNN3_IM_FMT_BGR888	BGR888 format, commonly used by OpenCV.
RKNN3_IM_FMT_GRAY8	8-bit grayscale image format.
RKNN3_IM_FMT_YCbCr_420_SP	YCbCr 4:2:0 semi-planar format, often used by video codecs.
RKNN3_IM_FMT_YCrCb_420_SP	YCrCb 4:2:0 semi-planar format, often used by video codecs.

Enum Value	Description
RKNN3_IM_FMT_YCbCr_422_SP	YCbCr 4:2:2 semi-planar format, often used by video codecs.
RKNN3_IM_FMT_YCrCb_422_SP	YCrCb 4:2:2 semi-planar format, often used by video codecs.
RKNN3_IM_FMT_UNKNOWN	Unknown or unsupported image format.

5.4.16. rknn3_llm_task_type

LLM task type enumeration.

Enum Value	Description
RKNN3_LLM_TASK_GENERATE	Generation task.
RKNN3_LLM_TASK_EMBEDDING	Vectorization/embedding task.

5.4.17. LLMOutputCallbackState

Output callback stage enumeration.

Enum Value	Description
RKLLM_OUTPUT_CALLBACK_PREFILL_PROCESSING	Output callback processing in prefill stage.
RKLLM_OUTPUT_CALLBACK_PREFILL_FINISHED	Output callback finished in prefill stage.
RKLLM_OUTPUT_CALLBACK_DECODE_PROCESSING	Output callback processing in decode stage.
RKLLM_OUTPUT_CALLBACK_DECODE_FINISHED	Output callback finished in decode stage.

5.4.18. rknn3_im_proc_flag

Image processing flag enumeration, used to describe operations in image processing flows.

Enum Value	Description
RKNN3_IM_PROC_FLAG_NONE	No processing flag.
RKNN3_IM_PROC_FLAG_CROP	Enable crop operation.
RKNN3_IM_PROC_FLAG_RESIZE	Enable resize operation.
RKNN3_IM_PROC_FLAG_FILL	Enable fill operation for padding areas.
RKNN3_IM_PROC_FLAG_COLOR_SPACE_CONVERT	Enable color space conversion.
RKNN3_IM_PROC_FLAG_DECODE	Enable image decode (for encoded image formats).
RKNN3_IM_PROC_FLAG_ENCODE	Enable image encode (to encoded image formats).

5.4.19. rknn3_op_target_type

Backend target enumeration for custom operators.

Enum Value	Description
RKNN3_OP_TARGET_TYPE_CPU	Backend target is CPU.
RKNN3_OP_TARGET_TYPE_MAX	Maximum enum value.

5.4.20. rknn3_op_plugin_type

Custom operator plugin type enumeration.

Enum Value	Description
RKNN3_OP_PLUGIN_TYPE_POSTPROCESS	Postprocess plugin.
RKNN3_OP_PLUGIN_TYPE_CUSTOM_OP	Custom op plugin (currently not supported).
RKNN3_OP_PLUGIN_TYPE_MAX	Maximum enum value.

5.5. Data Structures

5.5.1. rknn3_input_output_num

Information structure for RKNN3_QUERY_IN_OUT_NUM.

Member	Data Type	Definition
n_input	uint32_t	Number of inputs.
n_output	uint32_t	Number of outputs.

5.5.2. rknn3_quant_info

Quantization information structure.

Member	Data Type	Definition
scale	float	Scale data.
zero_point	int32_t	Zero point data.

5.5.3. rknn3_tensor_attr

Structure to store query results for RKNN3_QUERY_INPUT_ATTR/RKNN3_QUERY_OUTPUT_ATTR.

Member	Data Type	Definition
index	uint32_t	Input parameter: The index of the input/output tensor. This needs to be set before calling rknn3_query.
name	char[RKNN3_MAX_NAME_LEN]	Name of the tensor.
n_dims	uint32_t	Number of dimensions.
shape	uint32_t[RKNN3_MAX_DIMS]	Array of effective dimensions.
aligned_size	uint64_t	Size of the tensor with aligned shape, in bytes.
n_stride	uint32_t	Number of strides.
stride	uint64_t[RKNN3_MAX_STRIDE_DIMS]	Stride of the tensor. For example, the stride of a 16x16 tensor is [16*16, 16, 1].
n_elems	uint32_t	Number of elements in the tensor.
dtype	rknn3_tensor_type	Data type of the tensor.
layout	rknn3_tensor_layout	Data layout of the tensor.
qnt_type	rknn3_tensor_qnt_type	Quantization type of the tensor.

Member	Data Type	Definition
qnt_info	rknn3_quant_info	Quantization information of the tensor.
core_id	int32_t	Core ID of the tensor buffer.

5.5.4. rknn3_sdk_version

SDK version information structure.

Member	Data Type	Definition
api_version	char[64]	Version of the RKNN3 API.
drv_version	char[64]	Version of the RKNN3 driver.

5.5.5. rknn3_core_mem_size

Information structure for weight and internal memory size per core.

Member	Data Type	Definition
core_id	int32_t	The physical NPU core ID the memory belongs to.
weight_size	uint64_t	Size of weight memory in bytes.
internal_size	uint64_t	Size of internal tensor memory in bytes.
reserved	uint8_t	Reserved field.

5.5.6. rknn3_custom_string

Custom string information structure.

Member	Data Type	Definition
string	char[1024]	Custom string, maximum length 1024 bytes.

5.5.7. rknn3_tensor_mem

Tensor memory information structure.

Member	Data Type	Definition
virt_addr	void*	Virtual address of the tensor buffer.
phys_addr	uint64_t	Physical address of the tensor buffer.
fd	int32_t	File descriptor of the tensor buffer.
buffer_id	uint64_t	Buffer ID of the tensor buffer, used for memory management.

Member	Data Type	Definition
offset	uint64_t	Offset of the memory.
size	uint64_t	Size of the tensor buffer.
flags	uint64_t	Flags for the tensor buffer, reserved field.
core_id	int32_t	NPU core ID.
priv_data	void*	Private data for the tensor buffer.
mem_type	rknn3_mem_type	Memory type of the tensor buffer.

5.5.8. rknn3_config

Control parameter structure for model loading.

Member	Data Type	Definition
priority	int32_t	Execution priority.
run_timeout	uint32_t	Execution timeout in milliseconds. 0 means no timeout.
run_core_mask	uint32_t	Core mask for model execution.
user_mem_weight	uint8_t	Whether weight memory is user-allocated.
user_mem_internal	uint8_t	Whether internal memory is user-allocated.
user_sram	uint8_t	Whether SRAM memory is user-allocated.
reserved	uint8_t	Reserved field.

5.5.9. rknn3_tensor

Represents an RKNN3 tensor structure containing memory and attribute information.

This structure contains the memory information and attributes for a tensor used in RKNN3 operations. It is the fundamental data structure for handling tensors in the RKNN3 runtime.

Member	Data Type	Definition
mem	rknn3_tensor_mem*	Memory information of the tensor.
attr	rknn3_tensor_attr*	Attributes of the tensor.

5.5.10. rknn3_allocation_info

Memory allocation information structure for an RKNN3 model.

This structure provides detailed memory allocation information for different memory types (command, weight, internal, KV cache) and their distribution across NPU cores.

Member	Data Type	Definition
core_id	int32_t	NPU core ID.
command_mem	rknn3_tensor_mem	Command memory information.
weight_mem	rknn3_tensor_mem	Weight memory information.
internal_mem	rknn3_tensor_mem	Internal memory information.
kvcache_mem	rknn3_tensor_mem	KV cache memory information.
reserved	uint8_t	Reserved field.

5.5.11. rknn3_shape_info

RKNN3 model tensor shape information structure.

This structure contains information about the shapes of input and output tensors for an RKNN3 model, including tensor attributes and shape configuration details.

Member	Data Type	Definition
shape_id	int32_t	Unique ID for this shape combination.
n_inputs	uint32_t	Number of input tensors.
input_attrs	rknn3_tensor_attr*	Array of input tensor attributes.
n_outputs	uint32_t	Number of output tensors.
output_attrs	rknn3_tensor_attr*	Array of output tensor attributes.

Member	Data Type	Definition
is_default	uint8_t	Whether this is the default shape.
reserved	uint8_t	Reserved field.

5.5.12. rknn3_shape_config

Configuration structure for dynamic shape settings.

This structure contains information about the shape combinations and the currently active shape for dynamic shape inference in an RKNN3 model.

Member	Data Type	Definition
n_shapes	uint32_t	Number of available shape combinations.
current_shape_id	int32_t	ID of the currently active shape configuration. A value of -1 indicates no active shape.

5.5.13. rknn3_llm_config

LLM configuration structure.

This structure contains the basic configuration required to initialize and run an LLM.

Member	Data Type	Definition
chat_template	char*	Chat template string.
vocab_size	uint32_t	Vocabulary size.
embedding_dim	uint32_t	Embedding dimension (usually equal to hidden size).
max_ctx_len	uint32_t	Maximum context length.
max_position_embeddings	uint32_t	Maximum position embedding length.
kvcache_store_method	rknn3_kvcache_store_method	KV cache storage method.
kvcache_dtype	rknn3_kvcache_dtype	KV cache data type.
kvcache_group_size	uint32_t	Group size for KV cache group quantization.
kvcache_residual_depth	uint32_t	Residual depth for KV cache.
model_type	char*	Model type string.
task_type	rknn3_llm_task_type	LLM task type.
reserved	uint8_t	Reserved field.

5.5.14. rknn3_init_extend

Device-specific initialization information structure.

This structure is used to specify device-specific parameters during the initialization of an RKNN3 runtime context, including the device ID and reserved space for future use.

Member	Data Type	Definition
device_id	char*	Input parameter, indicates which device to select. Can be set to NULL if only one device is connected.
reserved	uint8_t	Reserved field.

5.5.15. rknn3_node_mem_info

RKNN3 device node memory information structure.

This structure provides detailed memory information for each node in an RKNN3 device, including the total and available memory for allocation.

Member	Data Type	Definition
total	uint64_t	Total memory available on this node, in bytes.
free	uint64_t	Free memory available on this node, in bytes.

5.5.16. rknn3_dev_mem_info

RKNN3 device node memory information structure.

This structure provides detailed memory information for each node in an RKNN3 device, including the total and available memory for allocation.

Member	Data Type	Definition
node_num	uint32_t	Number of nodes in the device.
sys_total	uint64_t	Total system memory of the device, in bytes.
sys_free	uint64_t	Available system memory of the device, in bytes.
node_mem_info	rknn3_node_mem_info	Memory information for each node.

5.5.17. rknn3_device

Structure representing an RKNN3 device.

This structure contains information about a specific RKNN3 device, including its ID, type and memory information.

Member	Data Type	Definition
id	char[RKNN3_MAX_DEV_LEN]	Device ID.
type	char[RKNN3_MAX_DEV_LEN]	Device type.
mem_info	rknn3_dev_mem_info	Memory information of the device.

5.5.18. rknn3_devices

Structure containing RKNN3 device information.

This structure contains the number of available RKNN3 devices.

Member	Data Type	Definition
n_devices	uint32_t	Number of devices.
devices	rknn3_device	Device information.

5.5.19. rknn3_vocab_info

RKNN3 model vocabulary information structure.

This structure contains information about the vocabulary used in an RKNN3 model, including its size and special token IDs.

Member	Data Type	Definition
vocab_size	int	Size of the vocabulary.
special_bos_id	int[RKNN3_MAX_SPECIAL_BOS_ID_NUM]	ID of the special Begin-Of-Sequence (BOS) token
special_eos_id	int[RKNN3_MAX_SPECIAL_EOS_ID_NUM]	ID of the special End-Of-Sequence (EOS) token
n_special_bos_id	int	Number of special Begin-Of-Sequence (BOS) token
n_special_eos_id	int	Number of special End-Of-Sequence (EOS) token
linefeed_id	int	ID of the linefeed token.
skip_special_token	bool	Whether to skip special tokens during generation.
ignore_eos_token	bool	Whether to ignore the EOS token during generation.
reserved	uint8_t	Reserved field for future expansion.

5.5.20. rknn3_llm_extend_param

LLM extended parameters structure.

Member	Data Type	Definition
reserved	uint8_t	Reserved field.

5.5.21. rknn3_sampling_params

Defines the sampling parameters for an LLM instance.

Member	Data Type	Definition
top_k	int32_t	Top-K sampling parameter for token generation.
top_p	float	Top-P (nucleus) sampling parameter.
temperature	float	Sampling temperature, affects the randomness of token selection.

Member	Data Type	Definition
repeat_penalty	float	Penalty for repeating tokens in generation.
frequency_penalty	float	Penalty for frequent tokens during generation.
presence_penalty	float	Penalty based on a token's presence in the input.

5.5.22. rknn3_llm_param

Defines the parameters for configuring an LLM instance.

Member	Data Type	Definition
logits_name	char*	Name of the output logits. This field only needs to be explicitly set if the model has multiple outputs; otherwise, it can be NULL.
max_context_len	int32_t	Maximum number of tokens in the context.
sampling_param	rknn3_sampling_params	Sampling parameters for token generation.
vocab_info	rknn3_vocab_info	Vocabulary information.
extend_param	rknn3_llm_extend_param	Extended parameters.

5.5.23. rknn3_lora

Defines the LoRA (Low-Rank Adaptation) parameters used in model fine-tuning.

Member	Data Type	Definition
lora_name	char[RKNN3_MAX_NAME_LEN]	Name of the LoRA.
scale	float	Scaling factor to apply to the LoRA.

5.5.24. rknn3_kvcache_policy_param

Defines the parameters for the KV cache policy.

If the model contains a system prompt, n_keep and n_keep_aligned are ignored.

Member	Data Type	Definition
recurrent.n_keep	int64_t	Number of caches to keep when recurrent; ignored if the model has a system prompt.
recurrent.n_keep_aligned	int64_t	Aligned number of caches to keep when recurrent, aligned to kvcache_group_size.
reserved	uint8_t[64]	Reserved field.

5.5.25. rknn3_llm_multimodal_tensor

Represents multimodal input (e.g., text, image, audio, and video).
Top-Level Members

Member	Data Type	Definition
name	const char*	Name of this tensor.
prompt	const char*	Text prompt input.
tokens	int32_t*	Token input. Mutually exclusive with prompt input; users can provide either one. Note: 1. If both are provided, prompt input will be used by default. 2. When using token input, users must manually convert text and multimodal tags into tokens according to the required rules.
n_tokens	uint64_t	Number of tokens in the token input. This must be set when using token input.
enable_thinking	bool	Controls whether "thinking mode" is enabled.
image	struct	Image sub-structure.
audio	struct	Audio sub-structure.
video	struct	Video sub-structure.

Image Sub-Structure Members

Member	Data Type	Definition
image_embed	float16*	Embedding of the image (size: n_image * n_image_tokens * embedding_dim * sizeof(float16)).
n_image_tokens	uint64_t	Number of image tokens.
n_image	uint64_t	Number of images.
image_start	const char*	Start tag for image in multimodal input.
image_end	const char*	End tag for image in multimodal input.
image_content	const char*	Content tag for image in multimodal input.
image_width	uint64_t	Width of image.

Member	Data Type	Definition
image_height	uint64_t	Height of image.

Audio Sub-Structure Members

Member	Data Type	Definition
audio_embed	float16*	Embedding of the audio (size: n_audio * n_audio_tokens * embedding_dim * sizeof(float16)).
n_audio_tokens	uint64_t	Number of audio tokens.
n_audio	uint64_t	Number of audio.
audio_start	const char*	Start tag for audio in multimodal input.
audio_end	const char*	End tag for audio in multimodal input.
audio_content	const char*	Content tag for audio in multimodal input.

Video Sub-Structure Members

Member	Data Type	Definition
video_embed	float16*	Embedding of the video (size: n_video * n_frame_per_video * n_frame_tokens * embedding_dim * sizeof(float16)).
n_frame_tokens	uint64_t	Number of frame tokens.
n_frame_per_video	uint64_t	Number of frames per video.
n_video	uint64_t	Number of video.
video_start	const char*	Start tag for video in multimodal input.
video_end	const char*	End tag for video in multimodal input.
video_content	const char*	Content tag for video in multimodal input.
frame_width	uint64_t	Width of the video frame.
frame_height	uint64_t	Height of the video frame.

5.5.26. rknn3_llm_tensor

Represents a tensor for large language model operations.

This structure contains the essential components for processing language model embeddings, including the tensor name, embedding vectors, token IDs, and the number of tokens.

Member	Data Type	Definition
name	const char*	Name of this tensor.
prompt	const char*	Text prompt input, if input_type is RKNN3_LLM_INPUT_PROMPT.
embed	float16*	Pointer to the embedding vectors (size is n_tokens * hidden_size), if input_type is RKNN3_LLM_INPUT_EMBED.
tokens	int32_t*	Array of token IDs, if input_type is RKNN3_LLM_INPUT_TOKEN.
n_tokens	uint64_t	Number of tokens represented in the embedding.
enable_thinking	bool	Controls whether to enable "thinking mode".

5.5.27. rknn3_llm_input

Structure representing different types of LLM inputs via a union.

Member	Data Type	Definition
role	const char*	Message role: "user" (user input), "tool" (function result).
input_type	rknn3_llm_input_type	Specifies the type of input provided (e.g., token, embed, aux).
llm_input	rknn3_llm_tensor	Embedding vector if input_type is RKNN3_LLM_INPUT_EMBED; Token array if input_type is RKNN3_LLM_INPUT_TOKEN; uses the prompt field if input_type is RKNN3_LLM_INPUT_PROMPT.
multimodal_input	rknn3_llm_multimodal_tensor	Multimodal input if input_type is RKNN3_LLM_INPUT_MULTIMODAL.
aux_input	rknn3_aux_tensor	AUX input if input_type is RKNN3_LLM_INPUT_AUX. Currently, rknn3_aux_tensor is the same type as rknn3_tensor.

5.5.28. rknn3_llm_infer_param

LLM inference parameter structure, defining parameters for the inference process.

Member	Data Type	Definition
keep_history	int	Determines the history retention flag (1: keep history, 0: discard history).
max_new_tokens	int32_t	Maximum number of new tokens to be generated.
reserved	uint8_t	Reserved field for future expansion.

5.5.29. RKLLMResult

Represents the result of an LLM inference.

Member	Data Type	Definition
token_ids	int*	Pointer to the tokens generated by the LLM.
num_tokens	int	Number of generated tokens.

5.5.30. RKLLMCallback

The RKLLMCallback structure contains callback functions for LLM operations.

Member	Data Type	Definition
result_callback	LLMResultCallback	Callback function for LLM return results.
result_userdata	void*	User data for LLMResultCallback.
sampling_callback	LLMSamplingCallback	Optional: Use only for custom sampling.
sampling_userdata	void*	User data for LLMSamplingCallback.
tokenizer_callback	LLMTokenizerCallback	Optional: Use only for a custom tokenizer.
tokenizer_userdata	void*	User data for LLMTokenizerCallback.
embed_callback	LLMGetEmbedCallback	Optional: Use only for custom embedding retrieval.
embed_userdata	void*	User data for LLMGetEmbedCallback.
output_callback	LLMOutputCallback	Optional: Use only to retrieve output tensors.
output_userdata	void*	User data for LLMOutputCallback.
output_tensors	rknn3_tensor*	Output tensor array pointer.
n_output_tensors	uint32_t	Number of output tensors.

5.5.31. RKLLMRunState

The RKLLMRunState structure contains state information for an LLM run.

Member	Data Type	Definition
n_total_tokens	uint64_t	Total number of tokens processed so far.
n_max_tokens	uint64_t	Maximum number of tokens that can be processed.
n_decode_tokens	uint64_t	Number of tokens generated in the decoding phase.
n_prefill_tokens	uint64_t	Number of tokens processed in the prefill phase.
kvcache_policy	rknn3_kvcache_policy	KV cache policy.
n_loras_enabled	int32_t	Number of enabled LoRAs.
loras_enabled	rknn3_lora*	List of enabled LoRAs.

5.5.32. rknn3_custom_op_context

Custom operator context structure, managed by the framework. Users can store custom data via user_data.

Member	Data Type	Definition
rknn_ctx	rknn3_context	RKNN3 context handle.
priv_data	void*	Framework private data.
user_data	void*	User-defined data.

5.5.33. rknn3_custom_op

Custom operator structure, including operator metadata and callback function set.

Member	Data Type	Definition
op_type	const char*	Custom operator type name.
plugin_type	rknn3_op_plugin_type	Plugin type.
target	rknn3_op_target_type	Backend execution target.
version	uint32_t	Operator version number.
author	const char*	Author information.
description	const char*	Operator description.
init	int ()(*rknn3_custom_op_context*)	Initialization callback (optional).
prepare	int ()(*rknn3_custom_op_context*, rknn3_tensor, *uint32_t, rknn3_tensor*, uint32_t)	Preparation callback (optional).
compute	int ()(*rknn3_custom_op_context*, rknn3_tensor, *uint32_t, rknn3_tensor*, uint32_t)	Computation callback (required).
deinit	int ()(*rknn3_custom_op_context*)	De-initialization callback (optional).
get_output_num	int ()(*rknn3_custom_op_context*)	Get output number callback (postprocess plugin, optional).
get_attrs	int ()(*rknn3_custom_op_context*, rknn3_tensor_attr, *uint32_t, rknn3_tensor_attr*, uint32_t)	Get input/output tensor attributes callback (postprocess plugin, optional).

5.5.34. rknn3_im_rect

Image rectangle structure used to describe regions of interest in image processing.

Member	Data Type	Definition
x	int	X-coordinate of the upper-left corner.
y	int	Y-coordinate of the upper-left corner.
width	int	Width of the rectangle in pixels.
height	int	Height of the rectangle in pixels.

5.5.35. rknn3_im_metadata

Metadata structure associated with RKNN3 image memory, used for bookkeeping between host and device.

Member	Data Type	Definition
peer_im_mem_addr	uint64_t	Address of the image memory object on the peer (remote) side.

5.5.36. rknn3_im_mem

Image memory information structure. It describes an image buffer stored in device memory and is used by image-related interfaces such as `rknn3_im_mem_create` and `rknn3_im_cvt_color`.

Member	Data Type	Definition
width	int	Image width in pixels.
height	int	Image height in pixels.
stride	int	Stride of the image buffer in bytes.
format	rknn3_im_fmt	Image format, see 5.4.15. rknn3_im_fmt.
sync_to_host	bool	Whether image data should be synchronized back to the host.
data_mem	rknn3_tensor_mem*	Pointer to the underlying tensor memory.
metadata	rknn3_im_metadata	Extra metadata shared between host and device.

6. RKNN3 C API Change Log

Session API Changes from v0.3.0 to v0.4.0 (2025-11-03)

> Reference demo1: rknn3-runtime/examples/rknn3_session_test_demo
> Reference demo2: rknn3_model_zoo/examples/Qwen2_5_VL

1. Added new multimodal input types: "audio/video", supporting flexible input combinations and arbitrary input orders of various modalities.
    - When entering a prompt, explicit modality tag markers should be inserted. For example: `You need to do three things: 1.<audio>Transcribe this audio; 2.<image>Describe the content of this image; 3.<video>Describe the content of this video.`
    - The `rknn3_llm_multimodal_tensor` structure has been improved. For example, members like `input_tensor.n_image` have been adjusted to sub-structures like `input_tensor.image.n_image` for clearer expression.
2. Added "thinking mode" switch functionality.
    - Enable thinking mode by setting `enable_thinking = true` (the default is false).
3. Added support for configuring multiple "eos tokens" to control the end of inference.
    - The initialization of `params.vocab_info.special_eos_id` is updated to an assignment in the form of a list.
4. Added the `rknn3_dump_features` API to run the model layer by layer and export intermediate tensors as `.npy` files. Output tensors are allocated internally when not provided.

Main Changes from v0.4.0b0 to v0.5.0 (2025-11-29)

1. Added postprocess-related query commands and custom operator plugin mechanism.
    - Added `RKNN3_QUERY_POSTPROCESS_*` query items.
    - Added `rknn3_register_custom_ops_plugins` and related structures/enumerations.
2. LLM callback and configuration enhancements.
    - Added `rknn3_session_set_function_tools`.
    - Replaced the "last hidden layer callback" with `LLMOutputCallback` + `LLMOutputCallbackState`.
    - Added `model_type` and `task_type` fields to `rknn3_llm_config`.
    - Added recurrent parameters to `rknn3_kvcache_policy_param`.
3. Improved image/YUV-related interfaces.
    - Extended `rknn3_im_mem_create` parameters to support size/core_id/flags.
    - Added `rknn3_im_proc_flag` enumeration for describing image processing flows.

Main Changes from v0.5.0 to v0.5.0 (2025-12-22)

1. Added performance and memory profiling interfaces.
    - Added `rknn3_profile_ops` and `rknn3_profile_mem`.