第5章 嵌入式人工智能多媒体开发架构

本章视频资料

本章学习目标

- 了解嵌入式人工智能多媒体开发架构。
- 了解 OpenCV 和 FFmpeg 开发方法。
- 掌握设备内存和系统内存的同步方法。

本章首先介绍嵌入式人工智能多媒体开发架构，并以算能公司的 BM1684 芯片架构为例介绍嵌入式人工智能多媒体开发的基本要点，包括开发环境搭建、内部功能应用等，其次介绍当前最主流的多媒体开发工具——OpenCV 和 FFmpeg 及其使用方法，最后介绍嵌入式人工智能开发中的设备内存和系统内存的同步方法。

5.1 概　　述

嵌入式人工智能通过底层多媒体处理接口向算法侧提供 API，如 OpenCV、FFmpeg 和 BMCV 接口。这些接口用于进行视频、图像的编解码以及图像的基本处理（如色彩空间转换、尺度变换、仿射变换等）。嵌入式人工智能多媒体开发框架如图 5.1 所示。多媒体开发是嵌入式人工智能芯片的核心功能，编解码能力是评价嵌入式人工智能芯片能力的关键指标。

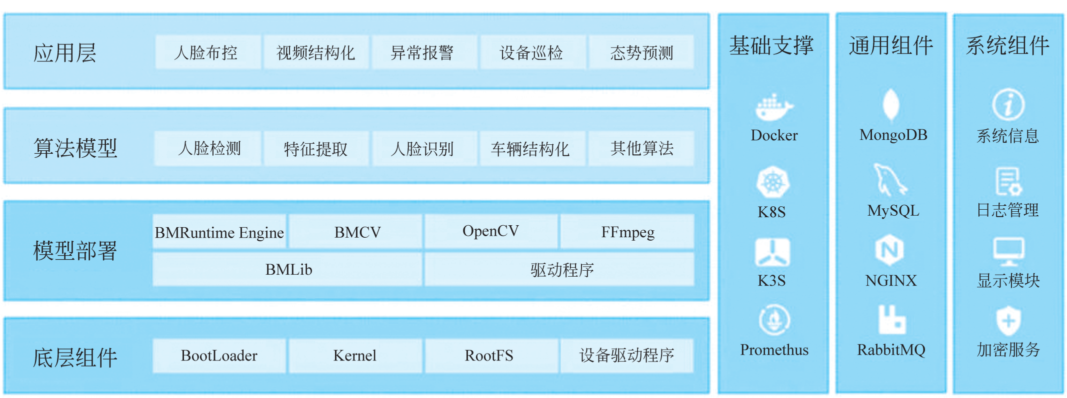


图 5.1　嵌入式人工智能多媒体开发框架

本章内容都以算能公司的 BM1684 芯片架构为例,如图 5.2 所示。BM1684 芯片作为比特大陆集团 BM 系列的最新版,聚焦于云端及边缘应用的人工智能推理,具有集成视频及图像编码和解码能力,实现了低功耗、高性能、全定制,可广泛应用于自动驾驶、城市大脑、智能政务、智能安防、智能医疗等诸多人工智能场景。其中视频子系统主要负责视频后处理和 MJPEG 的编解码工作。

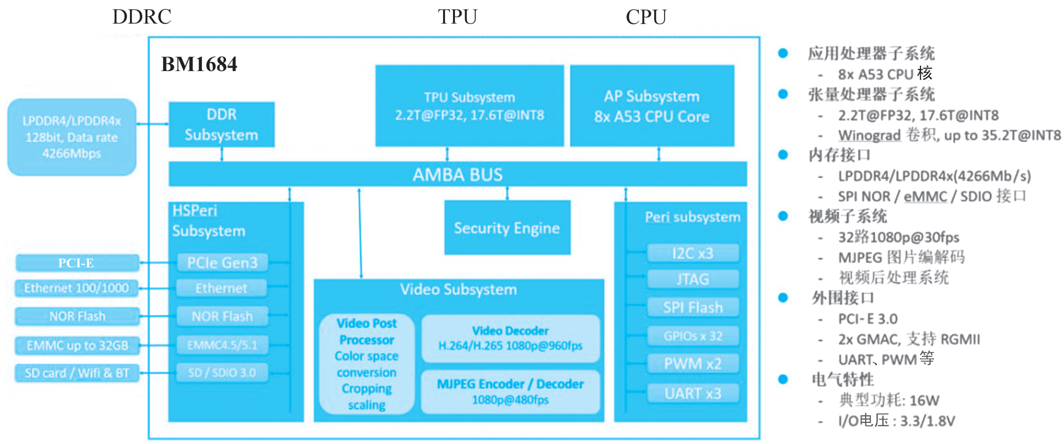


图 5.2 BM1684 芯片架构

5.1.1 开发架构

目前大部分嵌入式人工智能芯片基本都支持 OpenCV 和 FFmpeg,为用户提供统一的 OpenCV 或 FFmpeg 接口。嵌入式人工智能芯片通过修改底层,使其支持硬件加速等功能,能充分发挥芯片能力。

OpenCV 是目前视频图像处理领域应用最广的一个开源软件库,其目标是致力于提供易于使用的计算机视觉接口,从而帮助人们快速建立精巧的视觉应用[10]。

FFmpeg 作为音视频领域的开源工具,几乎可以实现所有针对音视频的处理。它不仅提供了音视频开发最基本的功能,使用范围也十分广泛,可实现包括视频采集、视频格式转换、视频抓图、给视频加水印等在内的功能。

以 BM1684 芯片为例,其多媒体框架的覆盖范围包括视频编码 VPU 模块、视频解码 VPU 模块、图像编码 JPU 模块、图像解码 JPU 模块、图像处理 VPP 模块。这些模块的功能都封装到 OpenCV 和 FFmpeg 开源框架中。其主要使用 OpenCV、FFmpeg 和 BMCV 这 3 个 API。在这 3 个框架中,OpenCV 框架擅长图像处理,各种图像处理算法最初都先集成到 OpenCV 框架中,而视频编解码通过底层调用 FFmpeg 实现;FFmpeg 框架擅长图像和视频的编解码,几乎所有格式都可以支持,只有是否能用硬件加速的区别;BMCV 专注于图像处理功能,且能使用 BM168x 硬件加速的部分。

OpenCV、FFmpeg 和 BMCV 这 3 个 API 在功能上的关系如图 5.3 所示。这 3 个框架之间可以灵活转换,不会发生大量数据复制所造成的性能损失。

(1) OpenCV 作为源计算机视觉库(计算机视觉工程师最常用的开源框架),封装了 FFmpeg 提供硬件加速的视频编解码接口和提供硬件加速的 JPEG 编解码接口,保留了原有的软件支持的图像处理功能。

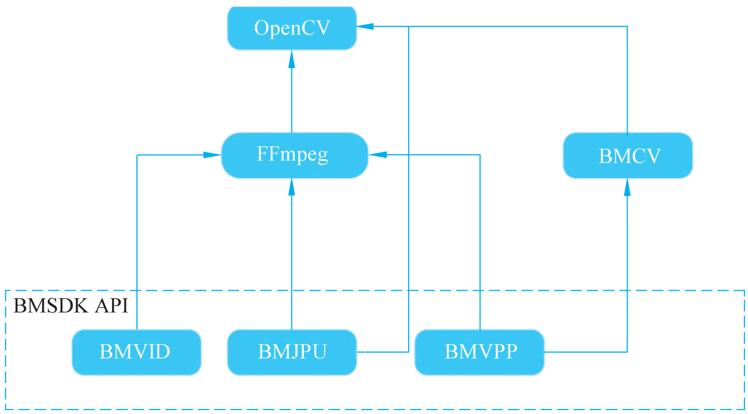


图 5.3　OpenCV、FFmpeg 和 BMCV 之间的关系

(2) FFmpeg 包含开源音视频及图像处理接口,提供硬件加速的 H.264 和 HEVC 视频编解码、JPEG 编解码、图像加速功能和所有软件支持的视频/图像编解码接口(即所有 FFmpeg 开源支持的格式)。

(3) BMCV 拥有自有图像处理加速接口,提供带硬件加速的图像处理功能。

注意,在视频编解码上,OpenCV 只是对 FFmpeg 接口的一层封装。使用 FFmpeg 时,控制粒度更细,灵活性更高,但开发代码量较大;而使用 OpenCV 接口时,代码更加简洁,开发速度更快,但可控程度较弱。

5.1.2　硬件加速

硬件加速是指通过硬件代替软件实现加速的方法,这充分利用了硬件本身处理速度快的性质,直接将代码存储在硬件中。目前人工智能、数据中心等技术热点对计算性能的需求向传统处理器提出了新的挑战,因此对采用 FPGA 或者 ASIC 进行硬件加速以获得更高能耗比的硬件加速计算方法有着非常迫切的需求。

硬件加速是嵌入式人工智能芯片的核心,利用芯片硬件提升复杂的编解码、图像处理能力。以 BM1684 芯片为例,硬件加速模块包括视频编解码模块、图像编解码模块和图像处理模块。

(1) 视频编解码。支持 H.264(AVC)和 HEVC 视频格式的硬件编解码加速,最高支持 HD(1080P)视频的实时编码和 4K 视频的实时解码。其编码的速度受到编码配置参数的影响,单芯片最高可以支持 2 路 HD 高清实时编码;而解码的速度则与输入视频码流的格式有关,一般单芯片可以支持 32 路 HD 高清实时解码。

(2) 图像编解码。支持 JPEG baseline 格式的硬件编解码加速。对于其他图像格式,包括 JPEG2000、BMP、PNG 以及 JPEG 标准的 progressive、lossless 等,均自动采用软解支持。图像硬件编解码的处理速度和图像的分辨率、图像色彩空间(YUV420p/422p/444p)有比较大的关系,一般而言,对于 1920×1080 分辨率、色彩空间为 YUV420p 的图像,单芯片硬件编解码可以达到 600fps 左右。

(3) 图像处理。BM1684 芯片中有专门对图像进行硬件加速处理的视频处理单元。支持的图像操作有色彩转换、图像缩放、图像切割(crop)、图像拼接(stitch)。VPP 不支持的复

杂图像处理功能则在 BMCV 接口中利用其他硬件单元作了特殊的加速处理。

5.1.3　工作模式

BM1684 芯片有两种工作模式：SoC 模式和 PCI-E 模式。如图 5.4 所示，SoC 模式是指用人工智能芯片中的处理器作为主控 CPU，可独立运行应用程序。PCI-E 模式是指以 PCI-E 板卡形态插到服务器主机上进行工作，作为一个硬件加速卡，其应用程序运行在服务器 CPU 上。

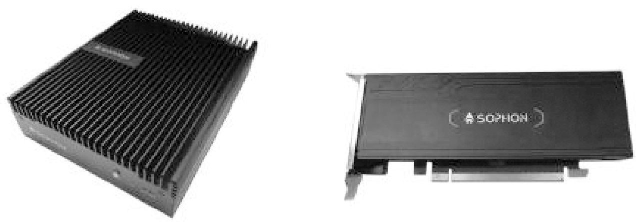


图 5.4　SoC 模式与 PCI-E 模式

图 5.5 与图 5.6 展示了上述两种模式的应用环境。

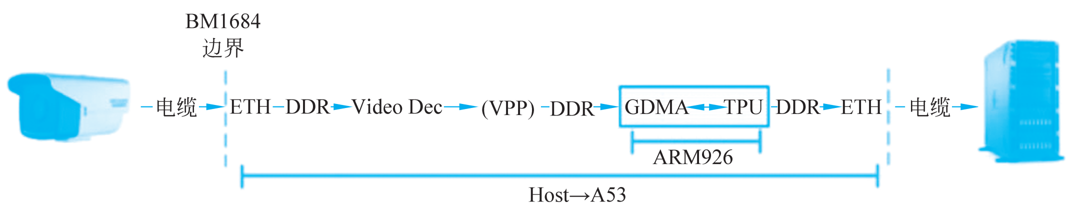


图 5.5　SoC 模式的应用环境

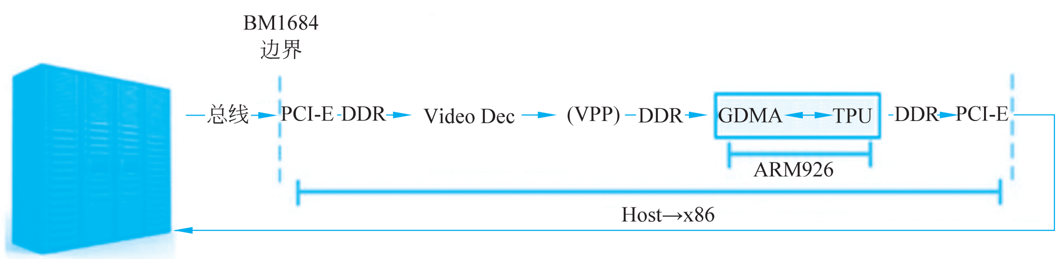
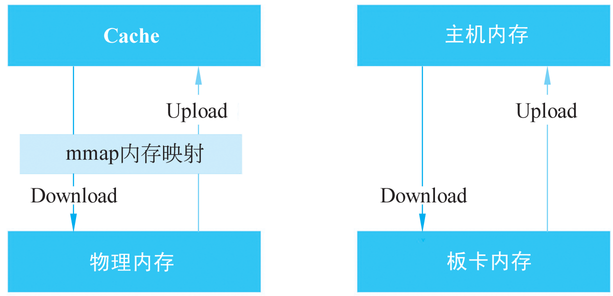


图 5.6　PCI-E 模式的应用环境

5.1.4　设备内存

当程序开发过程中对性能有进一步要求的时候，对于内存同步需要通过手动更加精准地控制，以避免一些不必要的内存同步导致的开销。此时需要理解 BM1684 芯片内部的内存同步控制机制，才能更好地用手动的方式进行内存控制。首先，在 SoC 模式下，设备内存和系统内存代表同一块内存，系统内存是操作系统对于物理内存的一个虚拟映射，在物理表现上其实仍然是同一块内存；而在 PCI-E 模式下，设备内存和系统内存就是真实的两块内存，称为主机内存和板卡内存。在这两种模式下，BM1684 芯片都对同步的方向作了设定，分别称为 Upload 方向和 Download 方向，如图 5.7 所示。

在 SoC 模式下，Upload 就是将物理内存的数据同步到 Cache 中，Download 就是将



(a) SoC模式下的内存同步　　(b) PCI-E模式下的内存同步

图 5.7　内存同步

Cache 中的数据同步到物理内存中，这里的两部分内存在主板上是同一块内存，同步的速度很快，开销很小。在 PCI-E 模式下，Download 指的是从主机内存将数据传输到板卡内存，Upload 指的是从板卡内存将数据传输到主机内存，这里的两部分内存代表两块不同的物理内存，因此同步时需要经过总线，同步速度较慢，开销大。

5.1.5　内存同步的时机

原则上，当需要处理的数据从软件处理切换到使用硬件模块加速的交界处，就要进行内存同步。下面分别介绍 3 种工具下的同步操作。

1. OpenCV 下的内存同步

在使用 OpenCV 工具时，BM1684 芯片采用以下方式进行内存同步：当调用 cv::BMCV::namespace 中的函数时，开发者根据要求设定 update 为 false 或者 true 即可进行内存同步。但是使用 OpenCV 的开发者需要注意的是：当使用 yuvMat 格式的数据进行开发的时候，最新的数据均在设备内存中；当使用 bgrMat 数据格式的时候，此时最新的数据均在系统内存中。开发者可以根据需要自行选择是否进行内存同步，以避免不必要的同步开销。OpenCV 下具体的手动内存同步的接口为 BMCV::downloadMat()或者 BMCV::upload()。

2. FFmpeg 下的内存同步

使用 FFmpeg 接口时，对于是否需要内存同步，FFmpeg 模块提供了相应的标志位，BM1684 芯片底层也对模块进行了改动。当开发者设定相应的标志位时，FFmpeg 会根据用户需要自动调用硬件加速，进行内存同步。

(1) 对于硬件解码模块的输入数据，开发者需要设置正确的 is_dma_buffer 标志位或者 hwupload filter，当设定为 true 时，即可开启硬件加速，启用内存同步。

(2) 对于硬件解码模块的输出数据，开发者需要设置正确的 zero_copy 标志位或者 hwdownload filter。

3. BMCV 下的内存同步

BMCV 的所有接口始终面向设备内存操作，所以当在 BMCV 提供的函数间进行数据交互时，开发者无须考虑内存同步。只有在以下情况的时候，才需要注意内存同步：

(1) 当前主机内存数据作为 BMCV 接口的输入。此时，当软件处理的数据需要用到 BMCV 模块中的函数，第一次调用 BMCV 函数接口时，要注意手动内存同步，将数据传输

到设备内存中，需要做一次 Upload 操作，将主机内存中的数据传输至板卡内存中去。

（2）当前内存数据作为 BMCV 接口的输出。此时主机需要获取板卡内存中处理好的数据，需要进行一次 Download 操作，进行内存同步，将数据从板卡内传输至主机内存中。

需要注意的是，在 SoC 模式下，当使用 Downlaod 同步物理内存到 Cache 内存中的时候，需要提前同步内存，这是由于 SoC 模式下的内存在物理上是同一块内存，当进行内存同步的时候，如果 Cache 中有数据，同步数据时也会将 Cache 中的数据再次写到物理内存中去，容易导致数据的覆盖，造成数据的破坏。而对于 PCI-E 模式来说就没有这个问题，PCI-E 模式在获得输出后直接同步内存即可。

5.1.6　手动内存同步的原因

当对数据进行处理时，最重要的就是数据不相互干扰，在 PCI-E 模式下，如果所有的数据均在设备内存上处理，不需要主操作系统的介入，此时不进行内存同步也没有关系；但是在大多数应用场景下，数据的处理都需要主操作系统的介入，必须进行内存同步。而内存同步又分手动内存同步和自动内存同步两种，对于 PCI-E 模式下的内存同步，自动同步会产生很多不必要的开销，例如简单的创建空间操作就会导致自动内存同步，数据在设备内存和主机内存之间不断地交互，对性能影响很大，所以要进行手动内存同步。而在 SoC 模式下，自动内存同步的开销很小，仅在同一块内存中进行覆写，使用自动同步没有问题。

5.1.7　内存同步示例

FFmpeg 是一个开源的跨平台音视频处理库，可以用于对音视频文件进行编码、解码、转码等操作。FFmpeg 支持多种格式的音视频文件，包括常见的 mp4、avi、flv 等格式。OpenCV 是一个开源的计算机视觉库，可以用于图像处理、目标识别、运动跟踪等方面。它支持多种编程语言，包括 C++、Python 等，并且具有良好的跨平台性能。这两个库在音视频处理和图像处理领域都非常流行，它们可以互相配合使用，例如，使用 FFmpeg 进行视频解码，然后使用 OpenCV 进行图像处理和分析。FFmpeg 和 OpenCV 都是常用的多媒体处理库，它们在不同的方面都有着优秀的表现。而有时候需要将它们配合使用的原因如下：

（1）FFmpeg 可以编码和解码多种视频格式，而 OpenCV 可以对视频图像进行分析和处理。因此，当需要从视频文件中提取图像进行进一步处理时，可以使用 FFmpeg 进行视频解码，然后使用 OpenCV 对图像进行分析和处理。

（2）OpenCV 的图像处理能力非常强大，但是对于大规模的视频处理任务，OpenCV 可能无法满足需求。而 FFmpeg 可以通过多线程和 GPU 加速等技术提高视频处理的效率和速度。当需要处理大规模的视频数据时，可以使用 FFmpeg 进行视频解码，然后使用 OpenCV 进行图像处理。

（3）有些特殊的视频格式，如一些实时视频流，可能不支持 OpenCV 直接读取。这时候就需要使用 FFmpeg 进行解码，并将解码后的视频帧转换成 OpenCV 支持的格式，才能进行进一步的图像处理。

因此，将 FFmpeg 和 OpenCV 结合使用可以发挥它们各自的优势，在多媒体处理和图像处理方面取得更好的效果。算能公司为了更好地利用硬件计算资源，又自行开发了 BMCV 库，其作用和 OpenCV 一致，只是在程序的底层做了更多的硬件适配，能够更好地利

用硬件资源。

综合以上分析,开发者可以根据个人需求自行选择习惯的开发套件,例如选择 OpenCV 或者 BMCV,必要时也可以利用不同的开发套件进行转换。下面给出了几种场景下各工具相互转换的范例。

1. FFmpeg 转换为 OpenCV

本例当使用完 FFmpeg 后需要调用 OpenCV 接口。

```
AVFrame * picture;
...
/* 中间经过 FFmpeg 的一系列处理,例如 avcodec_decode_video2()或者 avcodec_receive_
frame(),然后将得到的结果转成 Mat */
...
/* card_id 为进行 FFmpeg 硬件加速解码的设备序号,在常规 Codec API 中,可以通过 av_dict
_set()的 sophon_idx 指定,也可以在 hwaccel 设备初始化的时候指定。SoC 模式下默认为
0 */
cv::Mat ocv_frame(picture,card_id)

/* 还可以通过分步方式进行格式转换 */
cv::Mat ocv_frame;
ocv_frame.create(picture,card_id);
...
/* 然后可以用 ocv_frame 进行 OpenCV 的操作,此时 ocv_frame 格式为 BM168x 扩展的 yuv_
mat 类型。如果后续想转成 OpenCV 标准的 bgr_mat 格式,可以进行下列操作。注意:这里就有内
存同步的操作,如果没有设置,FFmpeg 默认是在设备内存中的,如果 update=false,那么转换成
bgr 的数据也一直在设备内存中,系统内存中为无效数据;如果 update=true,则设备内存同步到
系统内存中。如果后续还是用硬件加速处理,可以使 update=false,这样可以提高效率。当需要
用到系统内存数据的时候,显式调用 bmcv::downloadMat()同步即可 */
cv::Mat bgr_mat;
/* 根据需要设定 update 进行内存同步 */
cv::bmcv::toMAT(ocv_frame, bgr_mat,update);
...
/* 最后 AVFrame *picture 会被 Mat 中的 ocv_frame()释放,因此不需要对 picture 进行 av
_frame_free()操作。如果希望外部调用 av_frame_free()释放图像,则可以加上 card_id=
card_id | UMatData::AVFRAME_ATTACHED,该标准表明 AVFrame 的创建和释放由外部管理 */
ocv_frame.release();
picture = nullptr;
```

2. OpenCV 转换为 FFmpeg

下面给出 OpenCV 接口使用 FFmpeg 接口的实例。

```
/* 创建 yuv Mat。如果 yuv Mat 已经存在,可以忽略此步。card_id 为 BM168x 设备序号,SoC
模式下默认为 0 */
AVFrame * f = cv::av::create(height, width,AV_PIX_FMT_YUV420P,NULL,0,1,NULL,
NULL,AVCOL_SPC_BT709,AVCOL_RANGE_MPEG,card_id);
cv::Mat image(f,card_id);
...
/* 进行一些 OpenCV 操作 */
...
```

```
AVFrame * frame = image.u->frame;
/* call FFmpeg API */
...
/* 注意：在 FFmpeg 调用完成前，必须保证 Mat image 没有被释放，否则 AVFrame 会和 Mat image 一起释放。如果需要将这两个的声明周期分离开来，则上面的 image 声明要改成如下格式 */
cv::Mat image(f,card_id | UMatData::AVFRAME_ATTACHED);
/* 这样 Mat 就不会接管 AVFrame 的内存释放工作 */
```

1）OpenCV 转换为 BMCV

下面给出 OpenCV 接口转换为 BMCV 接口的实例。

```
cv::Mat m(height, width, CV_8UC3, card_id);
...
/* OpenCV 操作 */
...
bm_image BMCV_image;
/* 这里 update 用来控制内存同步，是否需要内存同步取决于前面的 OpenCV 操作。如果前面的操作都是用硬件加速完成的，设备内存中就是最新数据，就没必要进行内存同步，如果前面的操作调用了 OpenCV 函数，没有使用硬件加速（5.3 节中提到了哪些函数采用了硬件加速），对于 bgr mat 格式就需要进行内存同步。也可以在调用下面的函数之前，显式调用 cv::BMCV::uploadMat(m)实现内存同步 */
cv::BMCV::toBMI(m,&BMCV image,update);
...
/* 使用 BMCV_image 就可以进行 BMCV 调用，调用期间注意保证 Mat m 不能被释放，因为 BMCV_image 使用的是 Mat m 中分配的内存空间，handle 可以通过 bm_image_get_handle()获得 */
...
/* 释放内存。必须调用此函数，因为在 toBMI 中创建了 bm_image，否则会有内存泄漏 */
bm image destroy(BMCV_image);
m.release();
```

2）BMCV 转换为 OpenCV

下面给出 BMCV 接口转换为 OpenCV 接口的实例。

```
bm_image BMCV_image;
...
/* 调用 BMCV API 给 BMCV_image 分配内存空间，并进行操作 */
...
Mat m_copy,m_nocopy;
/* 下面的接口将发生内存数据复制，转换成标准 bgr mat 格式。update 控制内存同步，也可以在调用完这个函数后用 BMCV::downloadMat()控制内存同步。csc_type 是控制颜色转换的系数矩阵，控制不同 YUV 色彩空间转换到 bgr */
cv::BMCV::toMAT(&BMCV_image, m_copy, update, csc_type);

/* 下面的接口将直接引用 bm_image 内存（nocopy 标志位为 true），update 仍然按照前面的描述，选择是否同步内存。在后续 OpenCV 操作中，必须保证 BMCV_image 没有释放，因为 mat 的内存直接引用自 bm_image */
cv::BMCV::toMAT(&BMCV_image, &m_nocopy,AVCOL_SPC_BT709,AVCOL_RANGE_MPEG,NULL.
-1,update, true);
/* 往下可以进行 OpenCV 操作 */
```

5.2 FFmpeg

5.2.1 FFmpeg 概述

FFmpeg 是一套可以用来记录、转换数字音频、视频,并能将其转换为流的开源计算机程序。它有非常强大的功能,包括视频采集、视频格式转换、视频抓图、给视频加水印等。FFmpeg 是在 Linux 平台下开发的开源项目,但它同样也可以在其他操作系统环境中编译运行,包括 Windows、macOS 等。

FFmpeg 框架的基本组成包括 AVFormat、AVCodec、AVFilter、AVDevice、AVUtil、swresample 和 swscale 模块库,如图 5.8 所示。

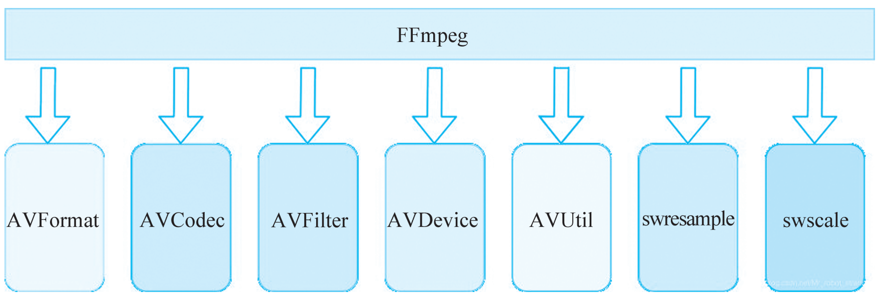


图 5.8 FFmpeg 框架的基本模块

(1) FFmpeg 的封装模块 AVFormat。FFmpeg 中的媒体封装格式主要由编译时是否包含该格式的封装库决定。一般通过在 AVFormat 中定制自己的封装格式以满足实际需求。而 AVFormat 包含目前多媒体领域中的绝大多数媒体封装格式,如 MP4、FLV、KV、TS 等文件封装格式,RTMP、RTSP、MMS、HLS 等网络协议封装格式。

(2) FFmpeg 的编解码模块 AVCodec。该模块中实现了目前多媒体领域绝大多数常用的编解码格式,既支持编码,也支持解码。AVCodec 除了支持 MPEG4、AAC、MJPEG 等自带的媒体编解码格式之外,还支持第三方的编解码器。如果要使用第三方编码方式,则需要安装对应的编码器。

(3) FFmpeg 的滤镜模块 AVFilter。该模块提供了一个通用的音频、视频、字幕等滤镜处理框架,它可以有多个输入和多个输出。

(4) FFmpeg 的设备管理模块 AVDevice。FFmpeg 的设备包括音频设备和视频设备。该模块提供了各种设备的输入输出接口。

(5) FFmpeg 的工具模块 AVUtil。该模块是 FFmpeg 中的基础模块之一,许多其他模块要依赖该模块实现基本的音视频处理操作。

(6) FFmpeg 的音频转换计算模块 swresample。该模块提供了高级别的音频重采样 API。例如,利用它可以操作音频采样、音频通道布局转换与布局调整。

(7) FFmpeg 的视频图像转换计算模块 swscale。该模块提供了高级别的图像转换 API。例如,利用它可以进行图像缩放和像素格式转换,常见操作是将图像从 1080P 转换成 720P 或者 480P 等,或者将图像数据从 YUV420p 转换成 YUYV,或者将 YUV 转换成 RGB 等图像格式。

在 FFmpeg 中也有可以供开发者使用的 SDK，是为各个不同平台编译完成的库，可以根据自己的需求使用这些库开发应用程序。

（1）libavcodec。包含音视频编码器和解码器。编解码库，封装了 Codec 层，但是有一些 Codec 是有自己的许可证的。FFmpeg 不会默认添加 libx264、FDK-AAC、Lame 等库，但是 FFmpeg 是一个平台，可以将其他的第三方 Codec 以插件的方式添加进来，为开发者提供统一接口。

（2）libavutil。包含多媒体应用常用的简化编程的工具，如随机数生成器、数据结构、数学函数等功能。它是核心工具库，是最基础的模块之一，其他模块都会依赖该库做一些基本的音视频处理操作。

（3）libavformat。包含多种多媒体容器格式的封装、解封装工具。它是文件格式和协议库，封装了 Protocol 层和 Demuxer、Muxer 层，使得协议和格式对于开发者来说是透明的。

（4）libavfilter。包含多媒体处理常用的滤镜功能。它是音视频滤镜库，包含了音频特效和视频特效的处理。在使用 FFmpeg 的 API 进行编解码的过程中，可以使用该库高效地为音视频数据做特效处理。

（5）libavdevice。用于音视频数据采集和渲染等功能的输入输出设备库。例如需要编译播放声音或者视频的工具 ffplay，就需要确保该库是打开的，同时也需要 libsdl 库的预先编译，该设备模块播放声音和视频都使用 libsdl 库。

（6）libswscale。用于图像缩放以及色彩空间和像素格式转换。该库用于图像格式转换，可以将 YUV 的数据转换为 RGB 的数据。

（7）libswresample。用于音频重采样和格式转换等。可以对数字音频进行声道数、数据格式、采样率等多种基本信息的转换。

（8）libpostproc。用于进行后期处理。当使用滤镜的时候，需要打开这个模块，滤镜会用到这个模块的一些基础函数。

作为音视频开发领域中最流行的开发库，FFmpeg 编解码主要有两种方式，分别是软解和硬解。通常在 Windows 下用 FFmpeg 不经过硬件加速进行编解码，称为软解。软解通过 CPU 执行算法进行解码，消耗 CPU 资源。通过设置，可以指定利用显卡的解码芯片对视频进行解码，则可称为硬解。

在智能多媒体应用中，FFmpeg 常用于从摄像头拉取视频流，并进行视频流的解码提取图像。FFmpeg 支持 RTSP 拉流，支持对 H.264 和 H.265 码流进行解码。FFmpeg 已经成为各大人工智能芯片厂家必然支持的开发库。厂家通常对外提供统一的、兼容的 FFmpeg 接口，在底层通过硬件对 FFmpeg 的编码或解码进行加速，这样既可以保证用户开发的一致性，又可以充分发挥人工智能芯片的硬件性能。

以解码过程为例，图 5.9 显示了利用 FFmpeg 进行解码的关键步骤。人工智能芯片厂家通常会基于开源 FFmpeg 封装自己的代码。

5.2.2　BM_FFmpeg

在 BM1684 芯片中，提供了 FFmpeg SDK 开发包，同时也提供了视频、图像等相关硬件加速模块。通过这些硬件接口，提供了如下模块：硬件视频编码器、硬件视频解码器、硬件