



# Arm<sup>®</sup> Streamline

Version 9.2

## Performance Advisor User Guide

**Non-Confidential**

**Issue 00**

Copyright © 2021–2024 Arm Limited (or its affiliates). 102009\_0902\_00\_en  
All rights reserved.



## Arm® Streamline Performance Advisor User Guide

Copyright © 2021–2024 Arm Limited (or its affiliates). All rights reserved.

### Release information

#### Document history

Issue	Date	Confidentiality	Change
0902-00	7 June 2024	Non-Confidential	New document for v9.2
0901-00	12 April 2024	Non-Confidential	New document for v9.1
0900-00	15 February 2024	Non-Confidential	New document for v9.0
0809-00	23 November 2023	Non-Confidential	New document for v8.9
0808-00	28 September 2023	Non-Confidential	New document for v8.8
0807-00	3 August 2023	Non-Confidential	New document for v8.7
0806-00	8 June 2023	Non-Confidential	New document for v8.6
0805-00	20 April 2023	Non-Confidential	New document for v8.5
0804-00	14 February 2023	Non-Confidential	New document for v8.4
0803-00	18 November 2022	Non-Confidential	New document for v8.3
0802-00	24 August 2022	Non-Confidential	New document for v8.2
0801-00	24 May 2022	Non-Confidential	New document for v8.1
0800-00	22 February 2022	Non-Confidential	New document for v8.0
0709-00	24 November 2021	Non-Confidential	New document for v7.9

### Proprietary Notice

This document is protected by copyright and other related rights and the use or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm Limited ("Arm"). No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted use, duplication, or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. Please follow Arm's trademark usage

guidelines at <https://www.arm.com/company/policies/trademarks>. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

## Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

## Product Status

The information in this document is Final, that is for a developed product.

## Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on <https://support.developer.arm.com>

To provide feedback on the document, fill the following survey: <https://developer.arm.com/documentation-feedback-survey>.

## Inclusive language commitment

### Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email [terms@arm.com](mailto:terms@arm.com).

# Contents

<b>1. Introduction.....</b>	<b>7</b>
1.1 Conventions.....	7
1.2 Useful resources.....	8
1.3 Other information.....	9
<b>2. Using Performance Advisor.....</b>	<b>10</b>
2.1 Overview of Performance Advisor.....	10
2.2 Performance report example.....	12
2.3 Performance Advisor workflows.....	15
2.4 API and device support.....	17
<b>3. Quick start guide.....</b>	<b>18</b>
3.1 Before you start.....	18
3.2 Connect Streamline to your device.....	20
3.3 Choose a counter template.....	21
3.4 Capture a Streamline profile.....	23
3.5 Generate a performance report.....	24
3.6 Setting performance budgets.....	27
3.6.1 Generating a report with per-frame performance budgets.....	27
3.7 Generate a custom report.....	28
3.8 Capturing slow frame rate images.....	32
<b>4. Running Performance Advisor in continuous integration workflows.....</b>	<b>34</b>
4.1 Generate performance reports automatically.....	34
4.2 Export performance data as a JSON file.....	36
4.3 Generate multiple report types.....	39
4.4 Generate a JSON diff report.....	40
<b>5. Adding semantic input to the reports.....</b>	<b>42</b>
5.1 Send and include annotations from application code.....	42
5.1.1 Send annotations from your application code.....	42
5.1.2 Include Streamline annotations in native applications.....	44
5.1.3 Include Streamline annotations in Unity applications.....	45

5.1.4 Include Streamline annotations in Unreal Engine applications..... 46

5.2 Specify a CSV file containing the regions..... 46

5.3 Clip unwanted data from the capture.....47

**6. Command-line options..... 49**

6.1 The Streamline-cli -pa command..... 49

6.2 The streamline\_me.py script options..... 53

# 1. Introduction

Learn how to use Performance Advisor to generate Android application performance reports from your Streamline data captures.

## 1.1 Conventions

The following subsections describe conventions used in Arm documents.

### Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: [developer.arm.com/glossary](https://developer.arm.com/glossary).

### Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
<i>italic</i>	Citations.
<b>bold</b>	Interface elements, such as menu names.  Terms in descriptive lists, where appropriate.
monospace	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
monospace <u>underline</u>	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments.  For example:  <pre>MRC p15, 0, &lt;Rd&gt;, &lt;CRn&gt;, &lt;CRm&gt;, &lt;Opcode_2&gt;</pre>
SMALL CAPITALS	Terms that have specific technical meanings as defined in the <i>Arm® Glossary</i> . For example, <b>IMPLEMENTATION DEFINED</b> , <b>IMPLEMENTATION SPECIFIC</b> , <b>UNKNOWN</b> , and <b>UNPREDICTABLE</b> .



Caution

We recommend the following. If you do not follow these recommendations your system might not work.



Your system requires the following. If you do not follow these requirements your system will not work.



You are at risk of causing permanent damage to your system or your equipment, or harming yourself.



This information is important and needs your attention.



A useful tip that might make it easier, better or faster to perform a task.



A reminder of something important that relates to the information you are reading.

## 1.2 Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at [developer.arm.com/documentation](https://developer.arm.com/documentation). Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Arm product resources	Document ID	Confidentiality
<a href="#">Arm community blogs</a>	-	Non-Confidential
<a href="#">Arm Streamline Target Setup Guide for Android</a>	101813	Non-Confidential
<a href="#">Arm Streamline Target Setup Guide for Bare-metal</a>	101815	Non-Confidential
<a href="#">Arm Streamline Target Setup Guide for Linux</a>	101813	Non-Confidential



Arm product resources	Document ID	Confidentiality
<a href="#">Arm Streamline User Guide</a>	101816	Non-Confidential
<a href="#">Get started with Performance Advisor Tutorial</a>	102478	Non-Confidential
<a href="#">Integrate Arm Performance Studio into a CI workflow</a>	102543	Non-Confidential
<a href="#">Optimization advice for graphics content on mobile devices</a>	102643	Non-Confidential

Non-Arm resources	Document ID	Organization
<a href="#">Configure on-device developer options</a>	-	<a href="#">Android Studio</a>
<a href="#">What is the ELK Stack?</a>	-	<a href="#">Elastic</a>

## 1.3 Other information

See the Arm website for other relevant information.

- [Arm® Developer](#).
- [Arm® Documentation](#).
- [Technical Support](#).
- [Arm® Glossary](#).

## 2. Using Performance Advisor

This section introduces the Performance Advisor tool and the workflows that it is designed to handle.

### 2.1 Overview of Performance Advisor

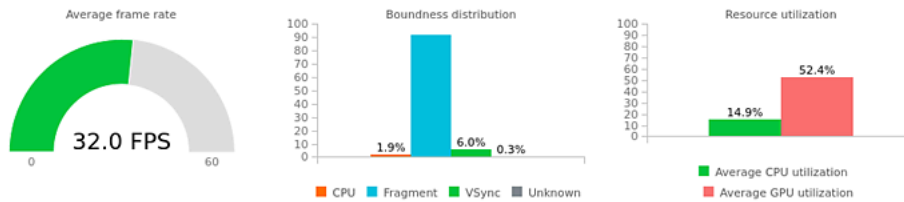
Performance Advisor analyzes performance data from your Streamline capture, and generates a report that shows how your application is performing on your mobile device.

The capture summary shows whether you are achieving your target frame rate, the distribution of time spent by each processing unit, and your CPU and GPU utilization.

**Figure 2-1: Example performance summary**

#### Capture summary ⓘ

🔍 You are hitting your performance target for 6% of the time within your application. For the frames below target you are predominantly fragment bound. Read our [optimization advice](#).

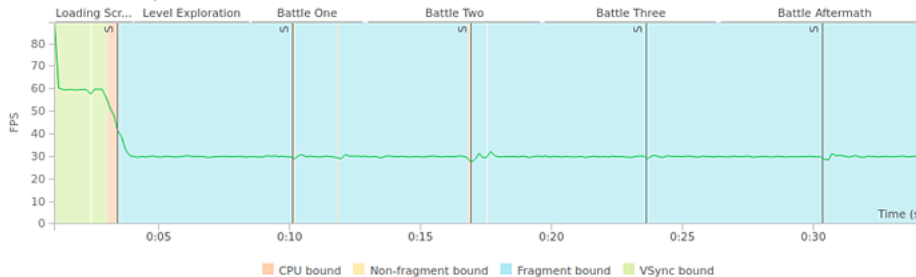


#### Region summary ⓘ

[Loading Screen](#) [Level Exploration](#) [Battle One](#) [Battle Two](#) [Battle Three](#) [Battle Aftermath](#)



#### Frame rate analysis ⓘ





This example performance summary includes regions, to help identify what is happening at different parts of the application. To enable regions and show region detail in your performance report, see [Send annotations from your application code](#) for more information.

To help you further understand how your application is performing over time, you can analyze key metrics shown on a series of charts:

### **Overdraw per pixel**

Identify problems caused by transparency or rendering order, by monitoring the number of times pixels are shaded before they are displayed.

### **Draw calls per frame**

To identify CPU workload inefficiencies, check the absolute number of draw calls per frame.

### **Primitives per frame**

See how many input primitives are being processed per frame, and how many of them are visible in the scene.

### **Pixels per frame**

See the total number of pixels being rendered per frame. This metric helps you to rule out problems caused by changes in the application render pass configuration. For example, extra passes for new shadow casters or post-processing effects.

### **Shader cycles per frame**

The total number of shader cycles per frame, broken down by pipeline, so that you can see which workloads are occupying the GPU.

### **GPU cycles per frame**

See how the GPU is processing non-fragment and fragment workloads, and whether the shader core resources are balanced.

### **GPU bandwidth per frame**

Monitor the distribution of GPU bandwidth, including the breakdown between reads and writes, so that you can minimize external memory accesses to save energy.

### **CPU cycles per frame**

See the consumption of CPU cycles per rendered frame. This metric helps you to validate improvements and regressions, which might not be visible in the CPU utilization charts.

Running the Performance Advisor report regularly enables you to get performance feedback throughout the development cycle. You can also integrate Performance Advisor in your performance regression workflows, by generating machine-readable JSON reports that you can import into other tracking systems.

Performance Advisor can identify scheduling issues that prevent you from achieving your target frame rate, and provide advice on how to resolve it. See [Generate a performance report](#) for more information.

## API support

Performance Advisor can instrument the OpenGL ES and Vulkan APIs, using layer drivers to supplement the Arm® Immortalis™ and Arm® Mali™ GPU performance counter data with software metrics and slow frame screenshots.

The layer drivers require the following API versions:

- OpenGL ES: 2.0 - 3.2
- Vulkan: 1.0 - 1.3

The layer drivers require the following Android versions:

- OpenGL ES: Android 10 and later
- Vulkan: Android 9 and later

Performance reports can still be captured on devices with older Android versions, but the application under test must manually generate the necessary frame boundary annotations. To learn how to add frame boundary annotations, see [Send annotations from your application code](#)

## Related information

[Performance report example](#) on page 12

[Quick start guide](#) on page 18

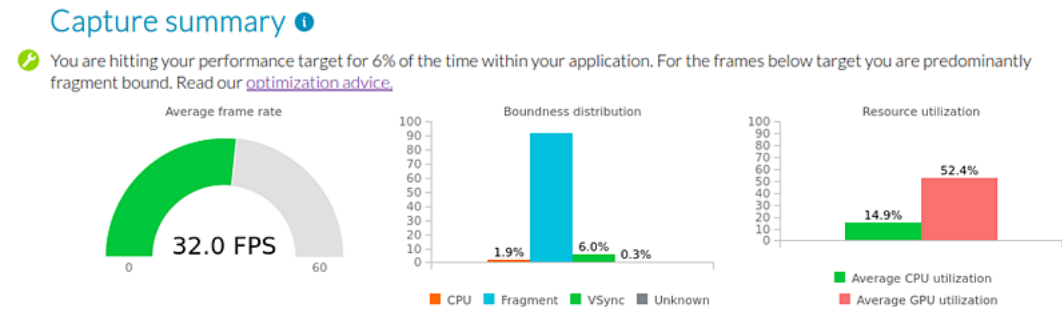
## 2.2 Performance report example

In this example, we will look at the charts in the Performance Advisor report to review the performance of an application. See how you can use the report to investigate problems with any scenes in your application that are not performing well.

We have generated a Performance Advisor report from a Streamline capture file.

### Report summary

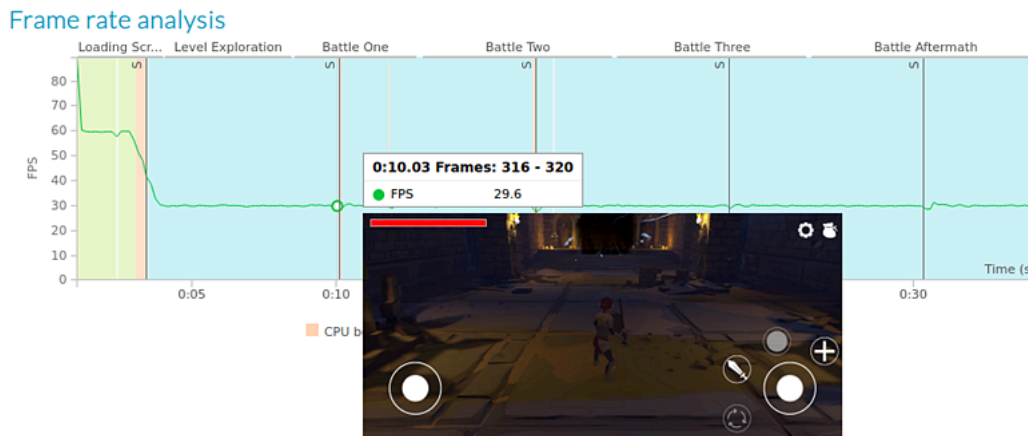
First look at the charts at the top of the report. These three charts provide a summary of how your application is performing for the duration of your capture. To identify any changes to your application throughout your development process, we recommend that you monitor these charts regularly.

**Figure 2-2: Example capture summary**

Here, we can see that the average frame rate for the capture is not achieving the configured target of 60fps. When we check the boundness distribution, we can see that the application is fragment bound. The utilization chart confirms that a graphical problem is causing this drop in frame rate.

## Analyze frame rate

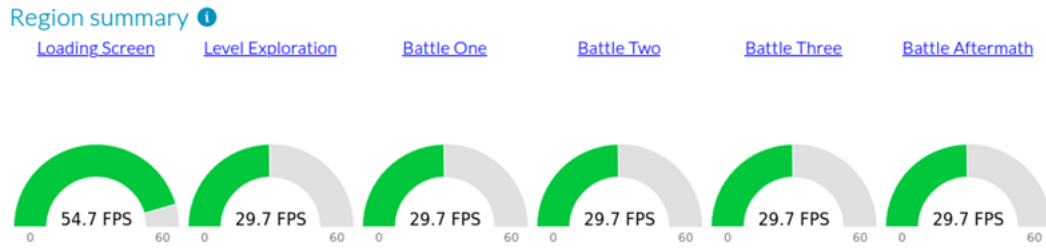
To see how the frame rate changes throughout the duration of your capture, check the **Frame rate analysis** chart.

**Figure 2-3: Analyze frame rate**

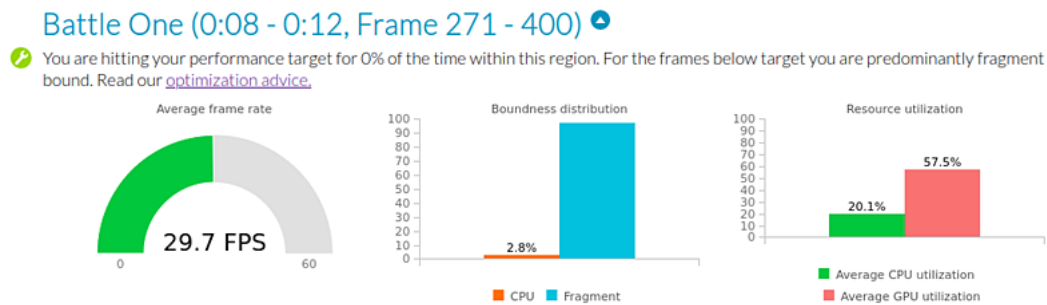
In this capture, we have used the `streamline_me.py` script to take a screenshot if the frame rate goes below 60fps. We have also specified a number of frames between screenshots to ensure that we do not capture too many images.

The majority background color of this chart is blue, indicating that the GPU in the device is struggling to process fragment workloads. We can also see that the frame rate has dropped below the target threshold of 60, so Performance Advisor has captured these frames. To see an image of the frame, hover the cursor on the screen capture icon ⓘ. In the image, you might be able to see which graphical element is causing the frame rate to drop.

Because we annotated our capture with regions, we can see the average frame rate for each region in the **Region summary** section.

**Figure 2-4: Region frame rate summary**

If a region requires more detailed analysis, we can select the chart title to go to the section for that specific region.

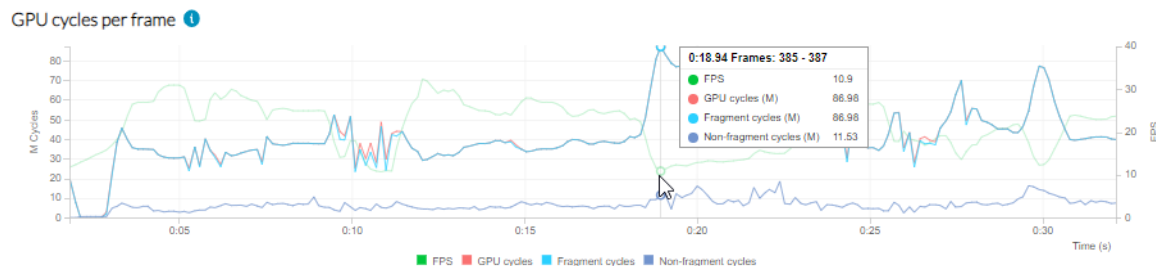
**Figure 2-5: Example region analysis**

To get a better understanding about what is happening in the application, we continue our analysis by looking at the GPU behavior metrics.

## Investigate GPU behavior

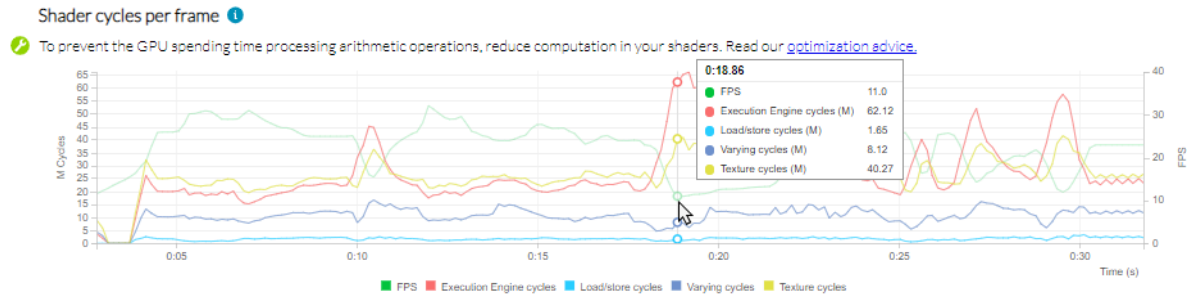
Scroll through the GPU behavior charts to find any strong correlation between the GPU metric and a drop in the frame rate. Performance Advisor provides advice above a chart where it finds a potential problem. You can also get further advice on optimizing your code by clicking the accompanying link to our developer website.

The **GPU cycles per frame** chart shows that the frame rate drops when the number of fragment cycles increases.

**Figure 2-6: GPU cycles chart**

The **Shader cycles per frame** chart shows that the drop in frame rate correlates with high numbers of execution engine cycles.

**Figure 2-7: Shader cycles chart**



This chart shows that the GPU is busy with arithmetic operations. We need to reduce the complexity of the shaders, and textures that we used. From here, we can click through to read [Optimization advice](#) about how to improve shader performance.

## Next steps

When you have identified a performance problem with Performance Advisor, use the other tools in the Arm® Performance Studio suite to explore your problem in more detail.

When sharing an HTML performance report, you must also share any high-resolution screenshot images that the report contains. Screenshot image files are output to `<report-filename>_image/`.

## Related information

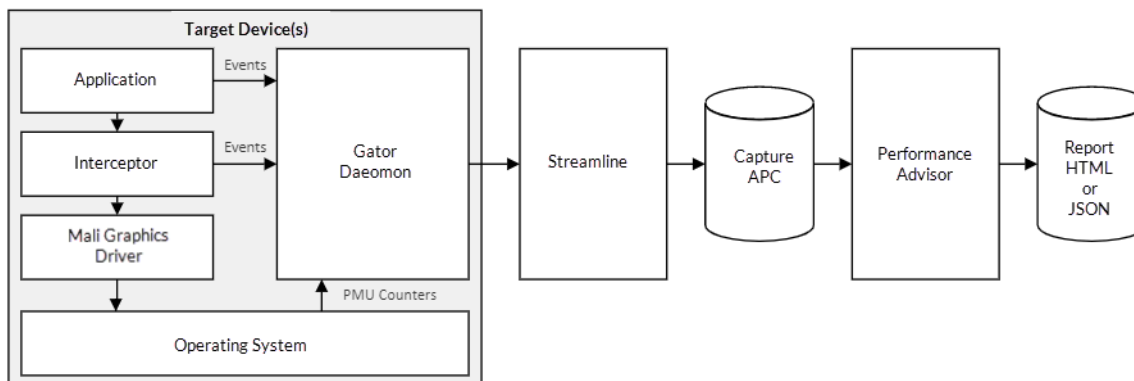
[Get started with Performance Advisor](#)

## 2.3 Performance Advisor workflows

You can use Performance Advisor with Streamline in interactive or automated workflows, so that you can manually debug a single problem, or set up regular reports to monitor performance over time.

### Interactive capture with Performance Advisor report

You can use Performance Advisor to assist with a manual debug session. Manually connect to a target and capture data using Streamline. Use Performance Advisor to post-process the dataset to provide an initial quick analysis.

**Figure 2-8: Interactive workflow.**

### Automated capture with Performance Advisor report

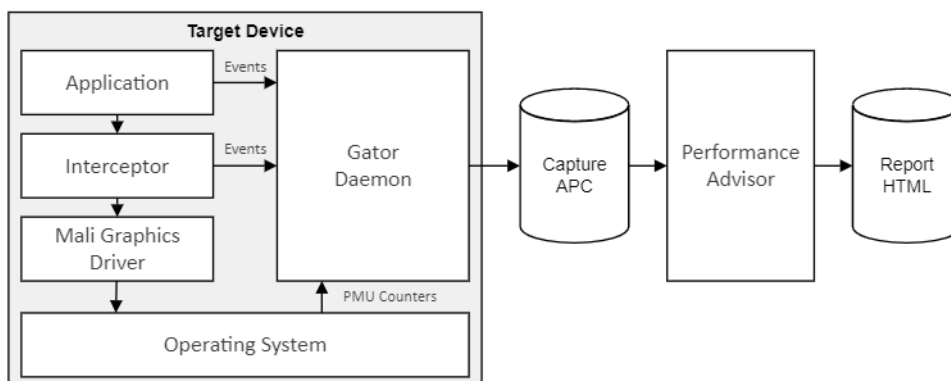
If your development team uses a CI system to merge daily code changes, you can integrate Performance Advisor and Streamline alongside this, to automate performance testing across multiple devices. Automatically generate summary reports in HTML for your team to analyze each morning, and export machine-readable JSON reports, so you can monitor performance over time. Build your own performance dashboards with any JSON-compatible database and visualization tool such as [ELK stack](#).

See [Running Performance Advisor in continuous integration workflows](#) for more information.



Note

The APC data file that the CI workflow creates is a full Streamline capture that you can import into the Streamline GUI. Arm recommends that you store the APC data file alongside other build artifacts. If Performance Advisor reports a problem, it is then immediately available for manual investigation in Streamline.

**Figure 2-9: Automated workflow.**

### Using Streamline and Graphics Analyzer for further deep-dive analysis

The Performance Advisor report shows where your application is causing a problem. You can then use the other tools in Arm® Performance Studio suite to investigate any problems in more detail.



## Streamline

Capture a profile of your application running on a mobile device and see where your system spends most of its time. Use interactive charts and comprehensive data visualizations to identify whether CPU processing or GPU rendering are causing any performance bottlenecks. For more information, see [Streamline](#) on the Arm Developer website.

## Graphics Analyzer

Graphics Analyzer enables you to evaluate all the OpenGL ES or Vulkan API calls your application makes, as it runs on an Android device. Explore the scenes in your game frame-by-frame, draw call-by-draw call, to identify rendering defects, or opportunities to optimize performance. For more information, see [Graphics Analyzer](#) on the Arm Developer website.

For more information about using Streamline for profiling graphical applications running on Arm® Mali™ GPUs, see the Arm Community blog [Accelerating Mali GPU analysis using Arm Mobile Studio](#).

## 2.4 API and device support

Describes the API support requirements for Performance Advisor, and a where to find the latest list of devices that Performance Advisor supports.

### API support

Performance Advisor can instrument the OpenGL ES and Vulkan APIs, using layer drivers to supplement the Arm® Immortalis™ and Arm® Mali™ GPU performance counter data with software metrics and slow frame screenshots.

The layer drivers require the following API versions:

- OpenGL ES: 2.0 - 3.2
- Vulkan: 1.0 - 1.3

The layer drivers require the following Android versions:

- OpenGL ES: Android 10 and later
- Vulkan: Android 9 and later

Performance reports can still be captured on devices with older Android versions, but the application under test must manually generate the necessary frame boundary annotations. To learn more about annotations, see [Send annotations from your application code](#).

### Related information

[Arm Performance Studio Release Notes](#)

## 3. Quick start guide

Performance Advisor runs on a capture file generated from Streamline. Follow the steps in this section when you are ready to perform an interactive capture.



If you already have the capture files, you can go straight to [Generate a performance report](#).

You can also watch a demonstration of the steps on the *Android profiling with Performance Advisor* video on [YouTube](#) or [Youku](#).

### 3.1 Before you start

Set up your computer and mobile device so that you can use Performance Advisor to analyze your applications.

#### Before you begin

Supplementary software metrics and slow frame screenshots require instrumenting the OpenGL ES and Vulkan APIs using layer drivers. The layer drivers require the following API versions:

- OpenGL ES: 2.0 - 3.2
- Vulkan: 1.0 - 1.3

and require the following Android versions:

- OpenGL ES: Android 10 and later
- Vulkan: Android 9 and later

For more information, see [API and device support](#).

#### Procedure

1. Download the studio package appropriate to your computer platform (Windows, Linux, or macOS):
  - Download Arm® Performance Studio from [Downloads](#)
  - Download Arm Development Studio from [Arm Development Studio Downloads](#)
2. Install your studio package:
  - On 64-bit Windows:

Arm Performance Studio is provided with an installer executable. Double-click the `.exe` file and follow the instructions in the **Setup Wizard**.

- On macOS:

Arm Performance Studio is provided as a `.dmg` package. To mount the package, double-click the `.dmg` package and follow the instructions. For easy access, the directory tree copies to the **Applications** folder on your local file system.

- On Linux:

Arm Performance Studio is provided as a `.tgz` tar archive. Use version 1.13, or later, of GNU `tar` to extract the archive to your preferred location:

```
tar xvzf Arm_Performance_Studio_<version>_linux.tgz
```

3. For macOS and Linux users only, you can edit your `PATH` environment variable so that you can run the Performance Advisor `streamline-cli -pa` command from any directory. On Windows, `PATH` is set automatically when you install Arm Performance Studio.

- On macOS, Arm provides a launcher file that opens the Terminal application and sets your `PATH` environment variable to include the path to the Streamline application. Navigate to the `<installation_directory>/streamline` directory and double-click the `streamline-cli-launcher` file. When your computer prompts you to allow Streamline to control the Terminal application, click **OK**.

Alternatively, you can edit the `/etc/paths` file to add the path to the `streamline` directory manually. The path is `/<installation_directory>/streamline/`.

- On Linux, you can edit your `PATH` environment variable to add the path to the Performance Advisor executable. Add the command `PATH=$PATH:/<installation_directory>/streamline` to the `.bashrc` file in your home directory, so that it is set whenever you initialize a shell session.
4. Install *Android Debug Bridge* (`adb`). The `adb` utility is in the Android SDK platform tools, which are installed as part of [Android Studio](#). Alternatively, you can download the latest version of `adb` from the [Android SDK platform tools site](#).
  5. Install Python 3.8 or later. Performance Advisor uses Python scripts to configure your device for data collection. You can download Python from the [Python Software Foundation](#).
  6. To run `adb` and `python3` commands from any directory, edit your `PATH` environment variable to add the paths to the location of the `adb` and Python executables.
  7. Connect your device to your computer through USB, then check that the device is set to [Developer Mode](#), and that **USB debugging** is enabled in **Settings > Developer options**. If your device prompts you to authorize connection to your computer, confirm the connection.
  8. To test the connection, enter the `adb devices` command in a terminal. If the connection is successful, the command returns your device ID:

```
adb devices
List of devices attached
ce12345abcdef1a1234    device
```

If your device is listed as unauthorized, restart USB debugging on your device, and ensure you accept the prompt on the device requesting you to authorize the connection to the computer.

9. Install a debuggable build of the application that you want to profile on the device:
  - In Android Studio, do one of the following actions:

- Create a build variant that includes `debuggable true` in the build configuration.
- Enable the `android:debuggable` setting in the application manifest file, as described in <https://developer.android.com/guide/topics/manifest/application-element>.
- In Unity, when building your application, select **Development Build** in **File --> Build Settings**.

## Next steps

- [API and device support](#)
- [Connect Streamline to your device](#)

## 3.2 Connect Streamline to your device

Arm® provides a Python script, `streamline_me.py`, that makes connecting to your device easy. Run the script so that Streamline can connect to your device, and collect data.

### Procedure

1. Open a command terminal on your host machine and navigate to the Performance Advisor installation directory, `<install_directory>/streamline/bin/android`.
2. Run the `streamline_me.py` Python script:

```
python3 streamline_me.py --lwi-mode counters
```

By default, the `streamline_me.py` script uses the light-weight interceptor to capture the OpenGL ES API. To capture the Vulkan API, use the `--lwi-api vulkan` option.



The `streamline_me.py` script expects to run from the installation directory. To create a directory containing the minimum set of files that is needed to make a capture, copy the following files from the Arm Performance Studio installation directory to a working directory:

- `<install_directory>/streamline/bin/android/streamline_me.py`
- `<install_directory>/streamline/bin/android/arm64/gatord`
- `<install-directory>/streamline/bin/android/<arm|arm64>/libGLESLayerLWI.so`
- `<install-directory>/streamline/bin/android/<arm|arm64>/libVkLayerLWI.so`

3. The script returns a numbered list of the Android package names for the debuggable applications that are installed on your device. Enter the number of the package you want to profile.  
The script identifies the GPU in the device, installs the daemon application, and waits for you to complete the capture in Streamline. Leave the terminal window open, as you must come back to it later to terminate the script.

#### 4. Launch Streamline:

- Open the Windows **Start menu**, navigate to the **Arm Performance Studio <version>** folder, and select the **Arm Streamline <version>** shortcut.
- On macOS, go to the <install\_directory>/streamline folder, and double-click the streamline.app file.
- On Linux, go to the <install\_directory>/streamline folder, and run the streamline file:

```
cd <install_directory>/streamline  
./Streamline
```

5. In the **Start** view, select your target device type. Then select your device from the list of detected targets, or enter the address of your target.
6. Android users only, select the package you want to profile from the list of packages available on the selected device.
7. TCP users only, optionally enter the details for any command you want to run on the application.

### Next steps

Choose a counter template. For more information about how to find and select a counter template, see [Choose a counter template](#).

### Related information

[Device connection issues](#)

## 3.3 Choose a counter template

Counter templates are pre-defined sets of counters that enable you to review the performance of both Arm® CPU and Arm GPU behavior. Choose the most appropriate template for the GPU in your target device.

### Before you begin

Follow the instructions in [Connect Streamline to your device](#) before you choose your counter template.

### Procedure

Select the counter template that you want to use to review performance of your CPU and GPU:

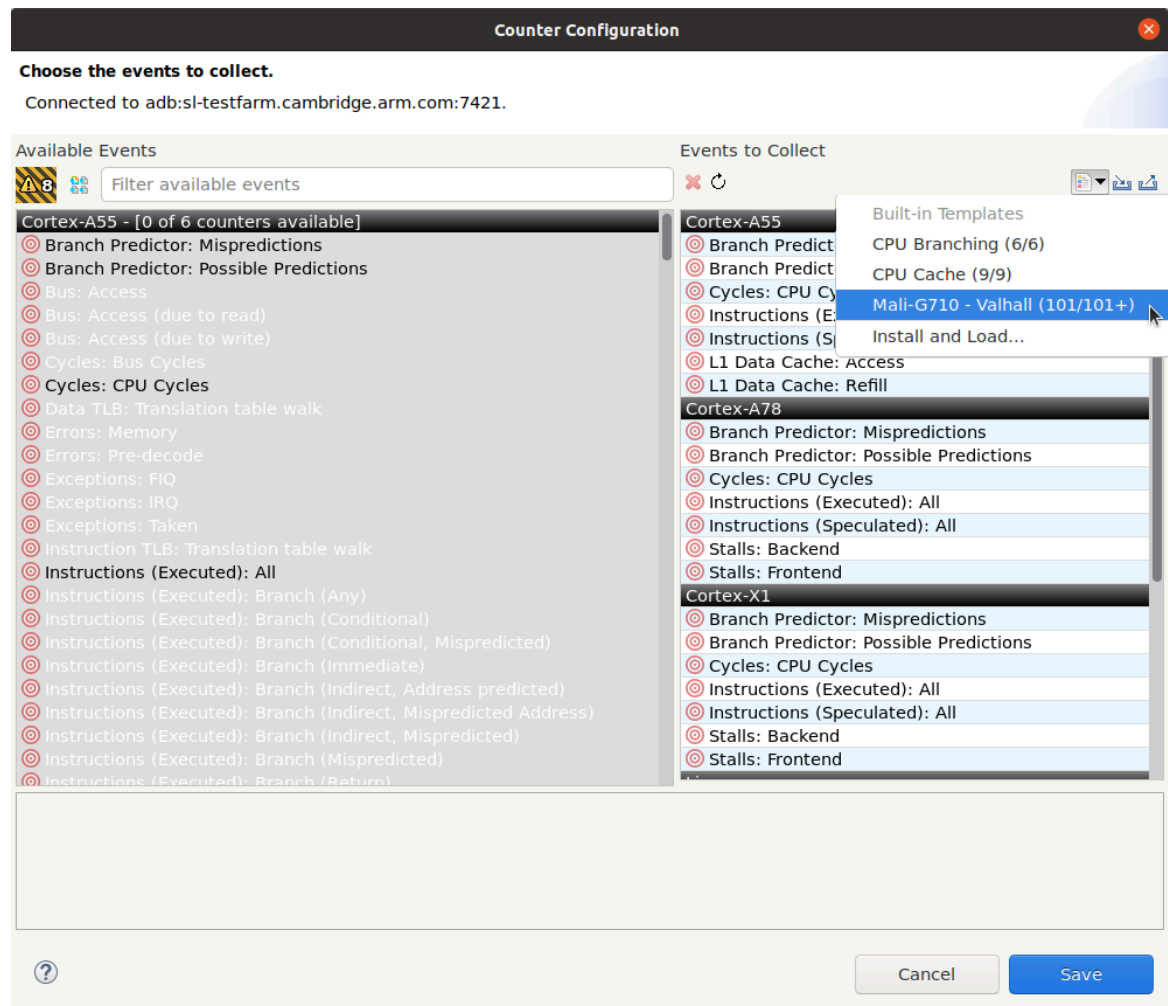
- To use the most appropriate counter template for your Arm GPU, select the **Capture Arm GPU profile** checkbox.
- To use the default counter template, which captures basic information such as CPU and memory usage, but no GPU data, clear the **Capture Arm GPU profile** checkbox.



If your device does not contain an Arm GPU, the **Capture Arm GPU profile** checkbox is disabled.

- To use a different counter template, select the **Use advanced mode** checkbox.
  1. Click **Select Counters**.
  2. Click **Add counters from a template**  to see a list of available templates.

**Figure 3-1: Templates available from the Select Counters dialog box.**



3. Select a counter template appropriate for the GPU in your device, then **Save** your changes.

The number of counters in the template that your device supports is shown next to each template. For example, here, 101 of the 101 available counters in the Arm® Mali™ template are supported in the connected device. Streamline notifies you if the target device does not support all the counters that are defined in the selected template.

Alternatively, to import an existing counter template, click **Install and Load**.

4. Optionally, click **Capture Settings** to set more capture options, including the sample rate and the capture duration (by default unlimited). See [Set capture options](#) in the *Arm Streamline User Guide*.

Streamline notifies you if the target device does not support all the counters that are defined in the selected template.

### Next steps

Capture a profile using Streamline. For more information about how to capture the behavior of your CPU and GPU performance using Streamline, see [Capture a Streamline profile](#).

## 3.4 Capture a Streamline profile

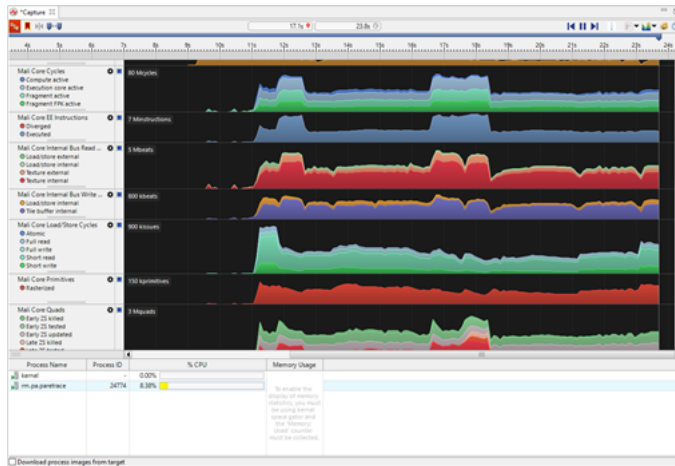
Start a capture session to profile data from your application in real time. When the capture session ends, Streamline automatically opens a report for you to analyze later.


### Before you begin

Before you capture a profile in Streamline, you must [Connect Streamline to your device](#) and [Choose a counter template](#).

### Procedure

1. In the **Start** view, click **Start capture** to start capturing data from the target device. Specify the name and location on the host for the capture file that Streamline creates when the capture is complete. Streamline then switches to **Live** view and waits for you to start the application on the device.
2. Start the application that you want to profile. The **Live** view shows charts for each counter that you selected. Below the charts is a list of running processes in your application with their CPU usage. The charts now start updating in real time to show the data that `gator` captures from your running application.

**Figure 3-2: Live view shows charts capturing data from your running application.**

- Unless you specified a capture duration, in the **Capture Control** view, click **Stop capture and analyze**  to end the capture. Streamline stores the capture file in the location that you specified previously, and then prepares the capture for analysis. When complete, the capture appears in the **Timeline** view.
- IMPORTANT:** Switch back to the terminal running the `streamline_me.py` script and press any key to terminate it. For a headless capture, the script kills all processes that it started and removes `gator` from the target. For a capture in Streamline, the script kills all processes that it started and removes `gator` from the target, but the `configuration.xml` file is retained.

## Next steps

- [Generate a performance report](#)
- To analyze performance with Streamline, see [Analyze your capture](#) in the *Arm Streamline User Guide*.

## 3.5 Generate a performance report

Generate an HTML performance report from an existing Streamline capture.

### Before you begin

To generate a report, you must first [Connect Streamline to your device](#), [Choose a counter template](#), and [Capture a Streamline profile](#).

### Procedure

- Open a terminal in the directory containing your APC file.





The APC file can be a zip file or an uncompressed .apc directory.

## 2. Run Performance Advisor using the following command:

```
streamline-cli -pa <filename>.apc [options]
```



For Performance Advisor versions up to and including 8.4, use the `pa` command instead.

To control how the `streamline-cli -pa` command runs, you can pass various options to it. See [The Streamline-cli -pa command](#) for detailed descriptions of all the available options. You can also add multiple command-line options to a file that you pass to the `streamline-cli -pa` command.

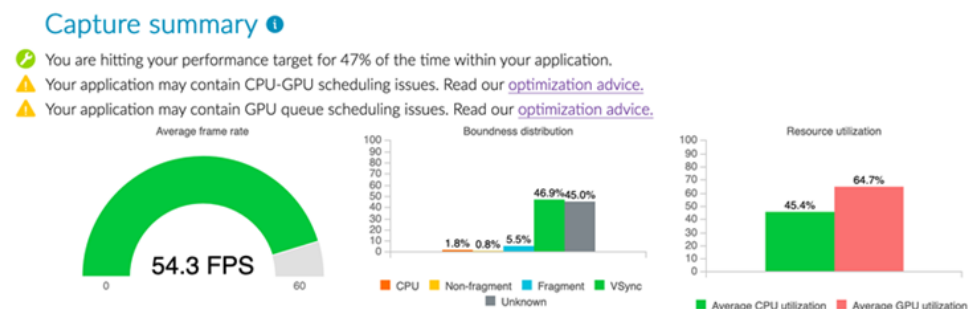
- You can use the `--help` option to list all the available command options for `streamline-cli -pa`.
- For example, to include build and device information in the report summary, include the `--build-name`, `--build-timestamp`, and `--device-name` command-line options.
- To show any CPU and GPU scheduling issues with your application, include the `--main-thread` option and specify the thread that you want to analyze:

```
--main-thread=<thread-name>
```



If any scheduling issues are detected, Performance Advisor shows an indicator at the top of the report.

**Figure 3-3: Scheduling indicators on the Performance Advisor report.**



- To check whether your application exceeds certain threshold values, include options for setting a per-frame budget.

## Results

Performance Advisor saves an HTML file to the current directory. Alternatively, you can specify a different directory using the `--directory` option. The file contains the results of the performance analysis, and links to advice on how to improve the performance.

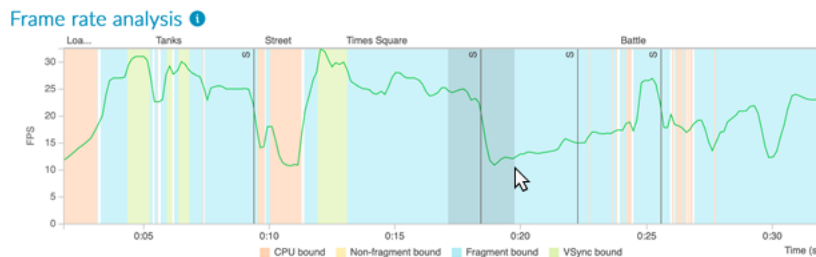


When sharing an HTML performance report, you must also share any high-resolution screenshot images that the report contains. Screenshot image files are output to `<report-filename>_image/`.

## Next steps

- The summary section shown at the top of the report is based on the duration of your capture. To take a closer look at a specific area of interest, click and drag the cursor over the area to select it.

**Figure 3-4: Zoom in to an area of interest.**



- Click anywhere on the chart when you are ready to go back to the original capture duration.
- You can zoom in to any line chart in the report in the same way, by clicking and dragging over the area of interest. When you zoom in on one chart, all other charts in the same section zoom in to the same point so you can easily compare them.
- If you set any per-frame budgets, a solid line appears on the relevant charts so you can check whether your application remains below it.
- To get help on overcoming graphics problems and optimizing your application, click the [Optimization advice](#) links on the report.

## Related information

[The Streamline-cli -pa command](#) on page 49

[Export performance data as a JSON file](#) on page 36

[Generate multiple report types](#) on page 39

[Optimization advice](#)

## 3.6 Setting performance budgets

As different target devices have different performance expectations, it is a good idea to set your own performance budgets based on the expected GPU performance.

If you know the top frequency for the GPU, and you have a target frame rate, you can calculate the maximum GPU cost per frame:

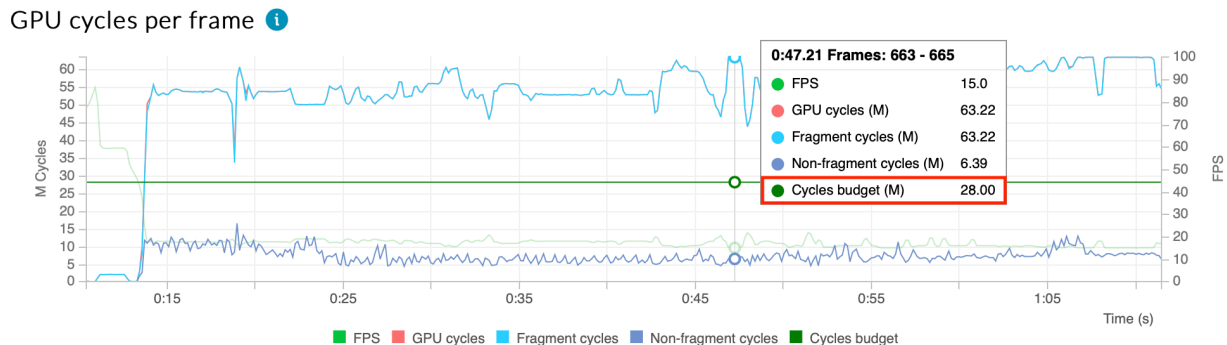
$$\text{GPU maximum frequency} / \text{frame rate} = \text{maximum GPU cycles per frame}$$

For example, if you want a minimum frame rate of 30fps on a device with a GPU with a maximum frequency of 940MHz, you can assume that the device can handle 31 million GPU cycles per frame.

$$940\text{MHz} / 30\text{fps} = 31.3\text{M}$$

When you generate Performance Advisor reports for this device, you can specify a maximum budget for GPU cycles per frame with the `--gpu-cycles-budget=<value>` command-line option to the `streamline-cli -pa` command. This budget is then shown on the GPU cycles per frame chart, making it easy to see when the application has exceeded the budget. Here, we set a budget of 28 million GPU cycles per frame but the number of fragment cycles is significantly higher than 28 million. This difference means the application is fragment bound.

**Figure 3-5: GPU cycles per frame with budget.**



All the per-frame charts in a Performance Advisor report can display a budget in this way.

### 3.6.1 Generating a report with per-frame performance budgets

To generate a Performance Advisor report where the charts show your own performance budgets for a device, use the relevant command-line options with the `streamline-cli -pa` command.

**Table 3-1: Relevant command options for reporting**

Command-line option	Budget
<code>--bandwidth-budget=&lt;value&gt;</code>	Threshold for read/write bytes.

Command-line option	Budget
--cpu-cycles-budget=<value>	Threshold for CPU cycles.
--draw-calls-budget=<value>	Threshold for draw calls.
--gpu-cycles-budget=<value>	Threshold for GPU cycles.
--overdraw-budget=<value>	Threshold for overdraw.
--pixels-budget=<value>	Threshold for pixels.
--primitives-budget=<value>	Threshold for primitives.
--shader-cycles-budget=<value>	Threshold for shader cycles.
--vertices-budget=<value>	Threshold for vertices.

For example:

```
Streamline-cli -pa mycapture.apc -gpu-cycles-budget=28000000
```

To make it easy to pass in several budgets, you can create a file containing your budget options. Pass this file directly to the `streamline-cli -pa` command when generating the report. See [The Streamline-cli -pa command](#) for detailed instructions.

## 3.7 Generate a custom report

To focus on the metrics that are most important to you, define which charts and series are included, and where they are shown, on the Performance Advisor report.

### Before you begin

- You must have a Streamline capture file. For help on creating a capture, see [Capture a Streamline profile](#).
- Your report can include a subset of Streamline charts that are suitable for processing as "per-frame" data, or it can include any series from any chart.

### Procedure

- Specify which charts or series you want to include in the report:
  - Use the `--chart-list-output` command-line option to generate a JSON custom report definition file that contains all possible charts that you can plot on the report. Remove the charts that you do not want to be displayed on the report. Fixed format charts, from the standard report, are displayed at the top of the report definition file generated by the `--chart-list-output` command-line option.
  - Alternatively, create your own JSON custom report definition file containing the names of the charts or series that you want to see on the report.



Some sample report definition files are available in the `examples` folder.

Example custom report definition file that contains Streamline charts:

```
{
  "groups": [
    {
      "title": "Memory Usage",
      "description": "This group shows the system memory usage charts.",
      "charts": [
        {
          "chart": "Mali Memory Bandwidth",
          "title": "Memory bandwidth per frame",
          "description": "This chart shows the distribution of GPU bandwidth. Minimize external memory access to reduce energy consumption.",
          "threshold": 100000000
        },
        {
          "chart": "Mali Core External Memory Reads",
          "title": "External memory reads per frame"
        }
      ]
    },
    {
      "title": "Texture usage",
      "charts": [
        {
          "chart": "Mali Core Texture Cycles"
        }
      ]
    }
  ]
}
```

Example custom report definition file that contains a custom chart defined using series expressions:

```
{
  "groups": [
    {
      "title": "Custom group",
      "description": "Custom group description",
      "charts": [
        {
          "id": "gpuUsagePerFrame",
          "title": "GPU cycles per frame",
          "series": [
            {
              "title": "GPU cycles",
              "expression": "$MaliGPUCyclesGPUActive"
            },
            {
              "title": "Fragment cycles",
              "expression": "$MaliGPUCyclesFragmentQueueActive"
            },
            {
              "title": "Non-fragment cycles",
              "expression": "$MaliGPUCyclesNonFragmentQueueActive"
            }
          ],
          "description": "This chart shows the distribution of GPU cycles across JS1 (vertex/tiler/compute) and JS0 (fragment) work queues.",
          "units": "Cycles"
        }
      ],
      "threshold": 100000000
    }
  ]
}
```

```
    ]
}
```

2. Enter information for the properties you require:

- The charts and series in your report must be contained within at least one group. The `groups` property enables you to group the charts in your report into different sections. You can add a heading for each section using the first `title` property. You can also add an `introduction` that appears on the report, and a `description`, which you reveal on the report using the drop-down icon.
- To add a Streamline chart to a custom report, enter the Streamline `chart` name exactly as it is shown in `--chart-list-output`. The `chart` name is the only mandatory property. To add more information about the charts in your custom report, add a `title` and a `description`.

You can also create a custom chart using series from any Streamline chart.

- To add a chart containing series to a custom report, enter the series name in the `title` property under `series`, and the Streamline expression name, exactly as it is shown in Streamline, in the `expression` property. The `series` and `expression` properties are mandatory. The `chart` name is not required.
- To stop Performance Advisor from performing per-frame processing on a series that already contains per-frame data, add `isPerFrame: true` to the corresponding series object in your custom report.
- To show how you are performing against your set per-frame budget, add a `threshold` value.



The `title`, `description`, and `threshold` are ignored for fixed-format charts, because the standard report format is used.

3. Run Performance Advisor using the following command:

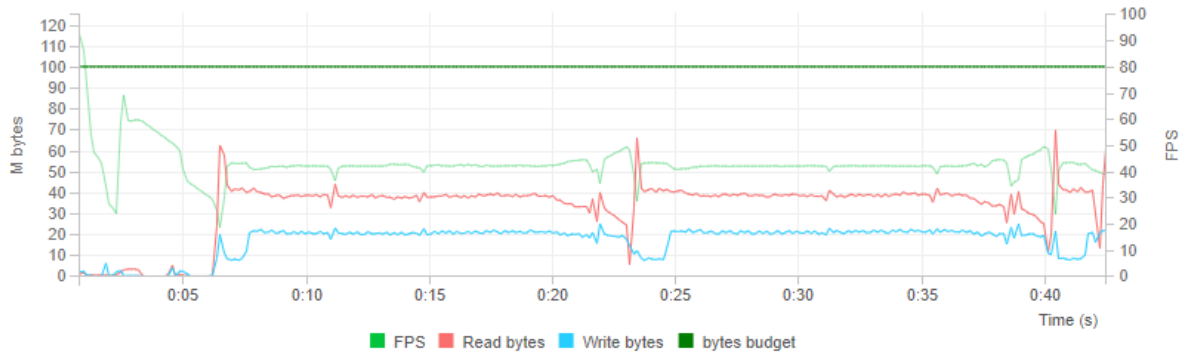
```
Streamline-cli -pa <filename>.apc --custom-report <path to configuration file>
[options]
```

Performance Advisor generates a custom report containing the charts specified in the custom report definition file, and any [The Streamline-cli -pa command](#) options specified. For example:

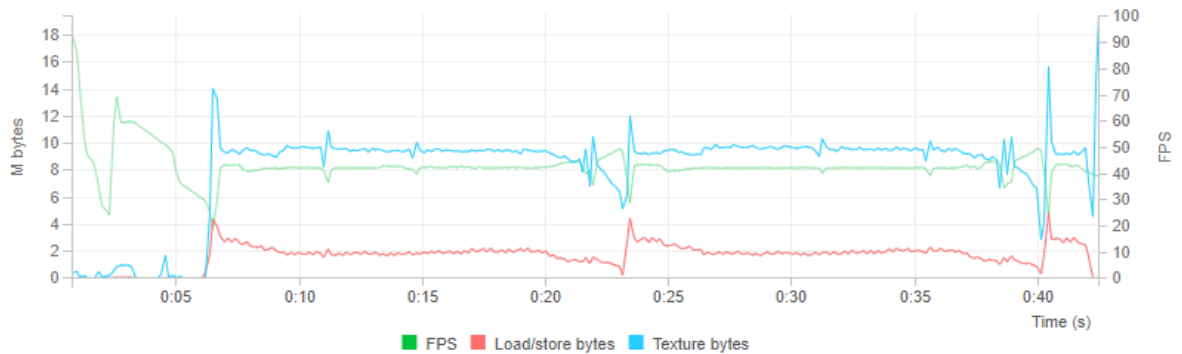
**Figure 3-6: Custom report displaying a group that contains two charts and a group that contains one chart.**

## Memory Usage

### Memory bandwidth per frame

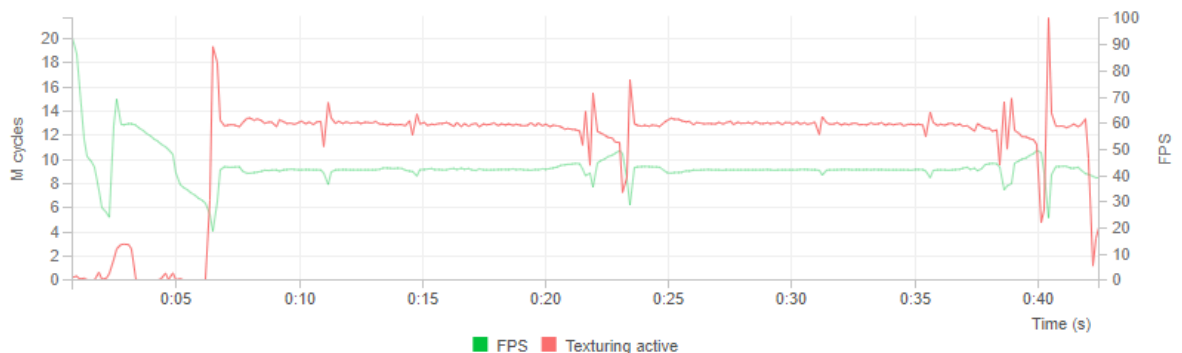


### External memory reads per frame



## Texture usage

### Mali core texture cycles per frame



## Results

Performance Advisor saves an HTML file to the current directory. Alternatively, you can specify a different directory using the `--directory` command-line option. The file contains the results of the performance analysis, and links to advice on how to improve the performance.



When sharing an HTML performance report, you must also share any high-resolution screenshot images that the report contains. Screenshot image files are output to `<report-filename>_image/`.

## 3.8 Capturing slow frame rate images

Use Performance Advisor to continuously monitor frame rate and trigger a frame screenshot capture when a slow part is detected.

### About this task

The Python script `streamline_me.py` enables you to capture data from your device using the Lightweight Interceptor (LWI) to monitor graphics API calls. This script is located in `<install_directory>/streamline/bin/android`.

### Procedure

1. In a terminal, navigate to `<install_directory>/streamline/bin/android`, where the Python script `streamline_me.py` is located.
2. Run the `streamline_me.py` script with the options you need for your frame capture. The script configures your device so that Performance Advisor can collect data from it.

For example, to capture a frame when the frame rate goes below 30fps, and allow at least 100 frames between captures:

```
python3 streamline_me.py --daemon <path_to_gatord> --lwi-fps-threshold 30 \  
--lwi-frame-gap 100 --lwi-mode screenshots \  
--lwi-out-dir <path_to_frame_captures_directory>
```

By default, the `streamline_me.py` script captures the OpenGL ES API using the lightweight interceptor. To capture the Vulkan API, use the `--lwi-api vulkan` option. See [The streamline\\_me.py script options](#) for details of all the available command-line options.



Enabling slow frame capture can affect application performance. You can reduce the impact by lowering the threshold FPS, increasing the minimum gap between captured frames, or disabling capture compression.

3. If there are multiple debuggable packages on your device, the script lists them. Enter the number of the package you want to analyze and follow the instructions to take a Streamline capture, as described in [Capture a Streamline profile](#).



When Streamline prompts you to save the capture file, do not save it to the frame captures directory that you specified in step 1. The contents of this directory are replaced when the frame capture images are written there.




4. To generate an HTML report, use the `streamline-cli -pa` command and specify the location where you saved the frame capture images in step 1. Optionally, specify a directory in which to save the HTML report, otherwise the HTML report is saved to the current directory.

```
Streamline-cli -pa <my_capture.apc> --frame-  
capture=<path_to_frame_captures_directory> \  
[--directory=<path_to_output_directory>]
```

You can use other options to specify metadata for your report, such as the build name, device name, and application name. See [The Streamline-cli -pa command](#) for all the available command-line options.

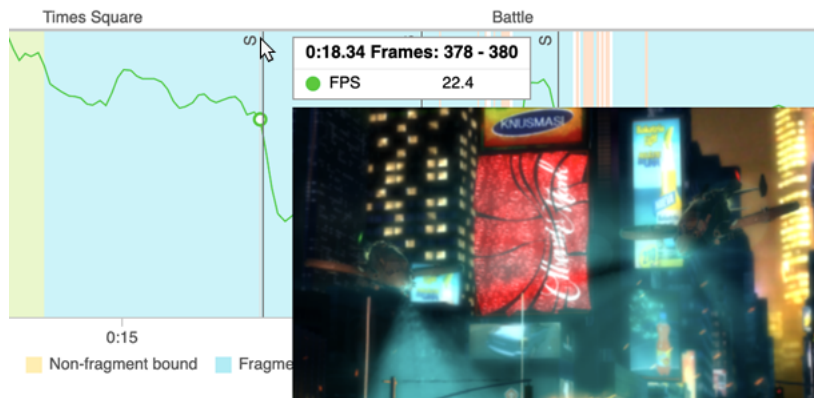
When a capture has multiple contexts, use the `-egl-context` argument to specify which context you want to display images from in the Performance Advisor report. If no context is specified, Performance Advisor displays images from the context that has the most images associated with it.

For more information about generating an HTML report, see [Generate multiple report types](#).

5. Open the HTML report in a browser.  
To see the captured frame, hover the cursor over the screen capture icon .

To view the captured frame in high resolution, use the middle-click on your mouse, also known as scroll click.

**Figure 3-7: Captured frame in HTML report.**



## 4. Running Performance Advisor in continuous integration workflows

Regular performance reports enable you to get instant feedback throughout your development cycle.

To automatically generate daily HTML or JSON reports that can help your team monitor how changes during the development cycle impact performance, integrate Performance Advisor into your Continuous Integration (CI) workflow:

- Human-readable HTML reports can give you a quick overview of your application performance, allowing you to monitor performance and identify new issues without having to manually open a Streamline capture.
- JSON reports provide a machine-readable format, allowing you to integrate metrics generated by Performance Advisor into your existing performance regression tracking systems. You can use JSON diff reports to compare two captures and highlight changes between them.

### 4.1 Generate performance reports automatically

If you use automated performance testing in Continuous Integration (CI), you can use performance reports to validate performance across multiple devices. For convenience, Performance Advisor can create performance reports using headless captures which do not require use of the host Streamline tool during data capture.

#### Before you begin

- Install Python 3.8, or later, to run the provided `streamline_me.py` script.
- Add the path to the `python3` directory to your `PATH` environment variable.
- Follow the instructions in [Configure counters](#) and [export a counter configuration file](#) to export a configuration containing the counters that you want to capture. Repeat this for each class of device you are going to profile.
- Use a CI tool such as Jenkins, TeamCity, or Buildbot to send the following instructions to the host machines for each device in your device farm.

#### Procedure

1. Change to the `<install_directory>/streamline/bin/android` directory, or copy the following files to your working directory:
  - `<install_directory>/streamline/bin/android/streamline_me.py`
  - `<install_directory>/streamline/bin/android/arm64/gatord`
  - `<install_directory>/streamline/bin/android/<arm|arm64>/libGLESLayerLWI.so`
  - `<install_directory>/streamline/bin/android/<arm|arm64>/libVkLayerLWI.so`
  - `configuration.xml`

2. Run the `streamline_me.py` script with the `--headless` option, specifying the package and activity to run, as well as passing any command line options needed. You must also specify the path to the counter configuration file:

```
python3 streamline_me.py \
--headless <path_to_output>/<output_file>.apc.zip \
--package <app.package.name> \
--package-activity <activity.name> \
--package-arguments <activity arguments> \
--lwi-mode counters \
--daemon <path_to_daemon>/gator \
--config <path_to_config_file>/configuration.xml
```

If you specify `--package-activity`, then the script automatically starts the activity when gator is ready to capture data. If you do not specify an activity, then you must start the activity manually in the next step. If you specify an activity, you can also specify command line arguments using `--package-activity` to pass an argument string. You must quote this string carefully to ensure it is correctly handled, for example:

```
# Argument value with no spaces
--package-arguments "--es fileName /sdcard/example/file.txt"

# Argument value with spaces and Bash quoting
--package-arguments "--es fileName \"/sdcard/example/file with spaces.txt\"""

# Argument value with spaces and PowerShell quoting
--package-arguments "--es fileName '/sdcard/example/file with spaces.txt'"'
```

By default, the `streamline_me.py` script uses the light-weight interceptor to capture the OpenGL ES API. To capture the Vulkan API, use the `--lwi-api vulkan` option. See [The streamline\\_me.py script options](#) for details of all the available command-line options.



If you built your application with Unity, include the Unity player activity in the application package name `<app.package.name>`, for example:

```
com.arm.mygame/com.unity3d.player.UnityPlayerActivity
```

3. If you did not use the `--package-activity` option and need to start the application manually. Wait for the script to prepare the device, and then start the application on the target device. For example:

```
adb shell am start -n <app.package.name>
```

4. To stop profiling, exit the application in one of the following ways:
  - Set your application test case to exit after a certain length of time.
  - Forcefully kill the application using:

```
adb shell am force-stop <app.package.name>
```

The Streamline capture file is saved to the location you specified with the `--headless` command-line option.



Instead of exiting the application, you can specify a `--headless-timeout <seconds>` value. This method is not ideal for test scenarios with variable performance.

5. Generate Performance Advisor reports in HTML and JSON formats:

```
Streamline-cli -pa <capture_filename.apc> -p <app.package.name> -d  
<output_directory> /  
-t html:<file_name>.html,json:<file_name>.json
```

For the full list of available command-line options, see [The Streamline-cli -pa command](#).

### Next steps

Push the HTML reports to a centrally visible location for your team to analyze each day. Push the JSON reports to any JSON-compatible database and visualization tool, such as [ELK Stack](#).

For more information, see [Integrate Arm Performance Studio into a CI workflow](#) on the Arm Developer website.

## 4.2 Export performance data as a JSON file

Generate a JSON report that you can import into other tools. Use reports from multiple test runs to track performance over time.

### About this task

JSON reports provide a raw data export that you can import into other tools, such as a NoSQL database, to compare different test runs. For example, you can track the average number of visible primitives per frame between builds.

### Procedure

1. Open a terminal in the directory containing your APC file.



The APC file can be a Streamline archive (.zip) or an uncompressed .apc directory.

2. Run Performance Advisor using the following command:

```
Streamline-cli -pa <capture.apc.zip> -p <app.package.name> -d <optional output  
dir> -t json
```

To change the output file name, append it to the `-t` argument using a colon:

```
-t json:your_file_name.json
```

### Example 4-1: JSON report

The JSON report output is packed by default, to make it compatible with most third-party database and visualization tools. If you want to view the data in a more human-readable format, use the `--pretty-print` option.

The following example shows part of a JSON report that was output with the `--pretty-print` option:

```
{
  "deviceInfo": {
    "build": null,
    "device": "Example board",
    "processors": "Cortex-A55 MP4, Mali-G72"
  },
  "allCapture": {
    "averageFrameRateFps": 19.4,
    "boundnessSplitPercentage": {
      "fragment": 0.0,
      "non-fragment": 0.0,
      "vsync": 0.0,
      "cpu": 98.5,
      "unknown": 1.5
    },
    "averageUtilizationPercentage": {
      "averageGpuUtilization": 19.0,
      "averageCpuUtilization": 62.7
    }
  },
  "fpsBoundness": {
    "frameRate": {
      "average": 19.4,
      "max": 21.1,
      "min": 17.9,
      "centiles": {
        "80": 20.0,
        "98": 21.1,
        "95": 20.7
      }
    },
    "vsync": {
      "target": 60,
      "percentageTimeUnderTarget": 100
    }
  },
  "overdrawPerPixel": {
    "overdraw": {
      "average": 0.3,
      "max": 0.4,
      "min": 0.1,
      "centiles": {
        "80": 0.4,
        "98": 0.4,
        "95": 0.4
      }
    }
  },
  "gpuUsagePerFrame": {
    "nonfragmentCycles": {
      "average": 1707767.6,
      "max": 2039630.8,
      "min": 770117.5,
      "centiles": {
        "80": 1917112.6,
        "98": 2039630.8,
        "95": 2039630.8
      }
    }
  }
}
```

```
    },
    "gpuCycles": {
      "average": 4157114.0,
      "max": 4897026.6,
      "min": 1587167.6,
      "centiles": {
        "80": 4649032.8,
        "98": 4897026.6,
        "95": 4897026.6
      }
    },
    "fragmentCycles": {
      "average": 2449346.8,
      "max": 2911080.0,
      "min": 608306.8,
      "centiles": {
        "80": 2857394.4,
        "98": 2911080.0,
        "95": 2911080.0
      }
    }
  },
  "drawCallsPerFrame": {
    "drawCalls": {
      "average": 456.0,
      "max": 456.0,
      "min": 456.0,
      "centiles": {
        "80": 456.0,
        "98": 456.0,
        "95": 456.0
      }
    }
  },
  "primitivesPerFrame": {
    "totalPrimitives": {
      "average": 290318.2,
      "max": 331233.8,
      "min": 114309.3,
      "centiles": {
        "80": 325304.5,
        "98": 331233.8,
        "95": 331233.8
      }
    },
    "visiblePrimitives": {
      "average": 89856.7,
      "max": 102210.2,
      "min": 34685.2,
      "centiles": {
        "80": 100151.9,
        "98": 102210.2,
        "95": 102210.2
      }
    }
  },
  "pixelsPerFrame": {
    "pixels": {
      "average": 4669783.4,
      "max": 5315129.7,
      "min": 3197000.8,
      "centiles": {
        "80": 5165539.5,
        "98": 5315129.7,
        "95": 5315129.7
      }
    }
  },
  ...
}
```



To aid writing parsers, JSON Schema definitions are provided in the `streamline/performance_advisor/json_schemas` directory.

## Related information

[The Streamline-cli -pa command](#) on page 49

[Generate a performance report](#) on page 24

[Generate multiple report types](#) on page 39

## 4.3 Generate multiple report types

Generate an HTML performance report and a JSON performance report from an existing Streamline capture.

### Before you begin

Before you can generate a report, you must have a Streamline capture file. For help on creating a capture, see [Capture a Streamline profile](#).

### Procedure

1. Open a terminal in the directory containing your APC file.



The APC file can be a zip file or an uncompressed `.apc` directory.

2. Run Performance Advisor using the following command:

```
Streamline-cli -pa <capture.apc.zip> -p <app.package.name> -d <optional output dir> -t html,json
```

To change the output file names, append each file name to the corresponding type argument using a colon:

```
-t html:your_file_name.html,json:your_file_name.json
```

### Results

Performance Advisor saves an each report output file to the current directory, or the directory specified in `-d <optional output dir>`. The files contains the results of the performance analysis, and links to advice on how to improve the performance.



Note

When sharing an HTML performance report, you must also share any high-resolution screenshot images that the report contains. Screenshot image files are output to `<report-filename>_image/`.

## Related information

[The Streamline-cli -pa command](#) on page 49

[Generate a performance report](#) on page 24

[Export performance data as a JSON file](#) on page 36

## 4.4 Generate a JSON diff report

To see how changes in your application affect performance, generate a diff report between two JSON reports to compare differences in performance metrics.

### Before you begin

You must have already generated two JSON reports. For help on exporting data as a JSON file, see [Export performance data as a JSON file](#).

### Procedure

Generate a JSON diff report using the following command:

```
Streamline-cli -pa --diff-report path/to/previous_json_report.json path/to/
current_json_report.json
```

This command subtracts the values in `previous_json_report.json` from the values in `current_json_report.json`.

### Results

Performance Advisor generates a file called `performance_advisor_diff-<timestamp>.json`, for example `performance_advisor_diff-210128-105937.json`. To specify a location for this file, use the `--directory` option.

Alternatively, to specify the filename of the JSON diff report, use the following command:

```
Streamline-cli -pa --diff-report-output mydiffreport.json path/to/
previous_json_report.json \
  path/to/current_json_report.json
```

To specify a location for the report, include the path in the filename or use the `--directory` option (see example).



Note

JSON diff reports can be validated against the JSON schema in `streamline/performance_advisor/json_schemas/pa_json_diff_report_schema.json`.



## Example 4-2: diff report locations

There are two ways to specify the location of the diff report that `--diff-report-output` generates.

- Include the path to the output directory with the filename:

```
Streamline-cli -pa --diff-report-output myoutputdir/mydiffreport.json  
previous.json current.json
```

- Specify the output directory with the `--directory` option:

```
Streamline-cli -pa --diff-report-output mydiffreport.json previous.json  
current.json \  
--directory myoutputdir
```

## 5. Adding semantic input to the reports

Performance Advisor can use semantic information that the application provides as key input data when generating the analysis reports.

The analysis reports support the use of region annotations to give context to the different frame ranges in a test scenario. Manually add these annotations into the application code. Alternatively, if manually adding annotations is not possible, or for quick debugging and extra analysis, specify a CSV file containing the regions. Give Performance Advisor the path to the CSV file using the `--regions` argument.

When you have added region annotations, your performance report includes a **Region summary** section, which contains charts showing the average frame rate for each region. You can also see the annotated regions in the **Frame rate analysis** chart, shown in the summary section of your report. See [Overview of Performance Advisor](#) for an example of regions shown on a report.

### 5.1 Send and include annotations from application code

You can send annotations from your application code using the Streamline annotations library. You can include the Streamline annotations library in native code, unity plug-in code, and unreal engine code.

#### 5.1.1 Send annotations from your application code

You can send annotations from your application code using the Streamline annotations library.

##### Procedure

1. Add frame or region boundaries depending on your use case:

##### **You want to avoid adding the lightweight interceptor to your application.**

The lightweight interceptor adds annotations to your Streamline capture that identify when frames begin and end. These annotations are then used by Performance Advisor to generate its analysis. If you avoid using the lightweight interceptor, Performance Advisor no longer knows when frames begin and end, and is not able to generate a report. Add frame boundaries yourself from your application code by calling:

```
ANNOTATE_MARKER_STR (FRAME_STR) ;
```

Where `FRAME_STR` takes the form of a monotonically incrementing frame number in the following regular expression format:

```
F (/d+)
```

For example:

```
F10  
F11  
F12
```



If you are using `streamline_me.py` to generate your capture, use the `lwi-mode=off` option to disable the lightweight interceptor.

### You want to specify a region from your application code.

Performance Advisor supports regions, which are subsets of time within the capture that represent a particular portion of the game. For example, a region can be a loading screen or a fight level scene within the capture. You can send this information from your application code by calling:

```
ANNOTATE_MARKER_STR(REGION_STR);
```

Where `REGION_STR` takes the form of:

```
Region Start <region name>  
Region End <region name>
```

For example:

```
ANNOTATE_MARKER_STR("Region Start Loading Screen");  
...  
ANNOTATE_MARKER_STR("Region End Loading Screen");
```

Performance Advisor creates a region in the report named "Loading Screen" for the time between the two markers.

Both of these approaches might create regions that are very short in time, resulting in a report that contains region data that is not useful. To configure your report to only include time regions that are longer than a minimum time threshold, use the `--region-report-min-length=time` argument.

Regions form a hierarchy where the parent starts before or at the same time as another overlapping region. You can configure your report to only include the breakout report of regions that are under a specified depth in this hierarchy by using the `--region-report-max-depth=depth` argument. All regions still appear in the **Frame Rate Analysis** summary graph regardless of hierarchy depth.

2. To enable the use of `ANNOTATE_MARKER_STR`, include the Streamline annotations library in your application using the relevant steps for your code:
  - [Native applications](#)

- [Unity applications](#)
- [Unreal Engine applications](#)

## Related information

[Include Streamline annotations in native applications](#) on page 44

[Include Streamline annotations in Unity applications](#) on page 44

[Include Streamline annotations in Unreal Engine applications](#) on page 45

## 5.1.2 Include Streamline annotations in native applications

Copy the necessary files into your project and include in the source files where you want annotations.

### Before you begin

The native C code for generating annotations in <mobile\_studio\_install>/streamline/gator/annotate.

### Procedure

1. Include the code in your project by completing one of the following sets of steps.
  - Copy the Streamline annotate file:
    - a. Copy `streamline_annotate.c` and `streamline_annotate.h` into your project directory.
    - b. Add the following line to any source file where you want to create annotations:

```
#include "streamline_annotate.h"
```

- Use a makefile:
  - a. Use `make` to compile a `libstreamline_annotate` library build using the makefile within the `annotate` directory.
  - b. Copy `libstreamline_annotate` into your projects directory.
  - c. Add the following line to any source file where you want to create annotations:

```
#include "libstreamline_annotate"
```

2. To start a thread to allow annotation for your program, add this line to one of your C files:

```
ANNOTATE_SETUP;
```

### 5.1.3 Include Streamline annotations in Unity applications

To generate Streamline annotations from user C# scripts, import the Performance Studio for Unity package and set up an assembly definition file. The assembly definition enables you to easily remove use of the package from release builds of your application.

#### About this task

The Performance Studio for Unity package is compatible with Arm® Mobile Studio version 2023.5 and earlier.

#### Procedure

1. Open the package manager in Unity.
2. Click **+** in the toolbar and select **Add package from git URL**.
3. Import the Performance Studio package from GitHub into your project.  
Use the import URL `https://github.com/ARM-software/mobile-studio-integration-for-unity.git`



See the [Performance Studio integration for Unity](#) project on GitHub for more information.

- 
4. If you do not have an `asmdef` file for scripts that reference the package API, create one.
  5. In the `asmdef` file, under `Assembly Definition References`, add `MobileStudio.Runtime`.
  6. In the `asmdef` file, under `Version Defines`, add a rule:
    - a) Set `Resource` to `com.arm.mobile-studio`.
    - b) Set `Define` to `MOBILE_STUDIO`.
    - c) Set `Expression` to `1.0.0`.

This rule makes Unity define `MOBILE_STUDIO` if the `com.arm.mobile-studio` package is present in the project and its version is greater than `1.0.0`.

7. In your code, wrap uses of the package annotation API with preprocessor guards:

```
#if MOBILE_STUDIO
// Package usage
#endif
```

#### Results

You can now generate Streamline annotations from your Unity C# scripts. You can also add and remove the package without breaking your project, which avoids errors in release builds.

### 5.1.4 Include Streamline annotations in Unreal Engine applications

Copy the necessary files into your project and include in the source files where you want annotations. You might require some additional libraries to compile the code.

#### Before you begin

You must have a C++ based project. Blueprint-based projects do not allow you to include external code.

#### Procedure

1. Follow the instructions in [Include Streamline annotations in native applications](#).



Some libraries that are required to compile the given code are not included with many compilers for Windows or within Microsoft Visual Studio. To download these packages within Visual Studio, complete the following steps:

2. Right-click on your project name within the **Solution Explorer** and select **Manage NuGet Packages for <project\_name>...**
3. Click **Browse**.
4. Select the **pthread** package.
5. Select all the checkboxes.
6. Click **Install**.

## 5.2 Specify a CSV file containing the regions

If manually adding annotations is not possible, or for quick debugging and extra analysis, specify a CSV file containing the regions and use the `--regions` argument.

Create a CSV file using the following format, where each region is on a new line:

```
Region Name,Start,End
```

`start` and `End` are a timestamp in milliseconds or a frame number followed by `ms` or `f`.

For example, specify a region that starts at 500ms and ends at 15000ms with:

```
Test Region,500,15000
```

Specify a region that starts at the 500th frame and ends at the 15000th frame with:

```
Test Region,500f,15000f
```

To set the start to the start of the capture, or the end to the end of the capture, use a \*. For example:

```
Test Region,*,15000
```

```
Test Region,5000f,*
```



Performance Advisor ignores the region if you use \* for both the start and the end, as this region is the whole capture.

Give Performance Advisor the path to the CSV file using the `--regions` argument.

## 5.3 Clip unwanted data from the capture

Specify the part of the capture that you want to include in the analysis report and discard the remaining data. For example, remove the application loading animation screens so they are not included in the report.

### About this task

You can specify the start and end time with one of the following:

- A timestamp in milliseconds.
- A region name with `:start` or `:end` appended to it.

### Procedure

1. Specify the start of the report with `--clip-start=<clipStartStr>`.



If you do not specify a start, the report starts from the beginning of the capture.

2. Specify the end of the report with `--clip-end=<clipEndStr>`.



If you do not specify an end, the report ends at the end of the capture.

### Example 5-1: Examples

- Clip the capture so the report starts at two seconds and ends at 15 seconds:

```
--clip-start=2000 --clip-end=15000
```

- Clip the capture so the report starts at the end of the region named "loading screen":

```
--clip-start="loading screen:end"
```

- Clip the capture so the report starts at the end of the region "level one loading screen" and ends at the start of the region "level two loading screen":

```
--clip-start="level one loading screen:end" --clip-end="level two loading  
screen:start"
```

### Related information

[The Streamline-cli -pa command](#) on page 49



## 6. Command-line options

This appendix explains the command-line options that are available for the `streamline-cli -pa` command and the `streamline_me.py` script.

### 6.1 The Streamline-cli -pa command

The `streamline-cli -pa` command runs Performance Advisor on a capture.



For Performance Advisor versions up to and including 8.4, use the `pa` command instead.

#### Syntax

To pass options directly to `streamline-cli -pa`, use:

```
Streamline-cli -pa [OPTIONS] <capture.apc>
```

To pass a list of options in a separate file to `streamline-cli -pa`, use:

```
Streamline-cli -pa <capture.apc> "@<options-file>"
```

#### Options

##### <capture.apc>

The path to the capture APC directory or zip file.

##### --centiles=int[,int...]

Comma-separated integer values specifying the percentiles to calculate for each data series. Default = 80,90,95.

##### --clip-end=clipEndStr

Specify the time that you want the report to end at. `clipEndStr` is the timestamp in milliseconds or the frame number followed by `f`. For example, `--clip-end=7000` ends the clip at 7000ms, or `--clip-end=7000f` ends the clip at the 7000th frame. Alternatively you can use the format `<region-name>:start` or `<region-name>:end` to use the start or end time of a region.

##### --clip-start=clipStartStr

Specify the time that you want the report to start from. `clipStartStr` is the timestamp in milliseconds or the frame number followed by `f`. For example, `--clip-start=500` starts the clip at 500ms, or `--clip-start=500f` starts the clip at the 500th frame. Alternatively you can use the format `<region-name>:start` or `<region-name>:end` to use the start or end time of a region.

**-d, --directory=path**

The output directory path for the reports.

**--egl-context=string**

Specify the context that you want to use for calculating the frame rate and to display images from in the Performance Advisor report. You can find the string for a specific context, which contains a '0x' prefixed hex value, in the screenshot filename, or by checking the Performance Advisor or Graphics Analyzer detail panels in Streamline.

**-f, --frame-capture=path**

The path to the frame captures directory.

**-h, --help**

Show command-line arguments and descriptions, and exit.

**-m, --main-thread=string**

The name of the main render thread to analyze.

**--mspf**

Display milliseconds per frame throughout the HTML report instead of FPS.

**--pretty-print**

Print the JSON output with whitespace, making it human readable.

**-p, --process=string**

A string specifying the process to inspect. The first part of the string is a regular expression that matches the process name. The second part states the desired process ID. The regex and process ID are separated by a '#' character. An example with the regex only: `--process=SampleApp`. An example with both regex and process ID: `--process=SampleApp#1234`. An example just specifying the process ID: `--process=#1234`.

**--[no-]progress**

Whether to display progress bars or not.

**-r, --regions=file**

Takes a CSV file containing custom regions to add to the report, where each line of the CSV file is of the format `regionName,start,end`. `start` and `end` are a timestamp in milliseconds or a frame number followed by `f`. For example, `regionName,500,7000` starts the region at 500ms and ends it at 7000ms. `regionName,500f,7000f` starts the region at the 500th frame and ends it at the 7000th frame. See [Specify a CSV file containing the regions](#).

**--region-report-min-length=time**

Minimum region length (in seconds) for a region to appear on the report. For example, `--region-report-min-length=0.5` removes any region that is less than 500ms long from your report.

**--region-report-max-depth=depth**

Regions form a hierarchy where the parent starts before or at the same time as another overlapping region. This option defines the maximum depth in that hierarchy allowed for a region to appear in the breakout report. All regions still appear in the **Frame Rate Analysis** summary graph. The first user region depth starts at 1.

**-t, --type=type[:file][,type[:file]...]**

A comma-separated list of report types, where the type is one of:

**json**

JSON CI report

**html**

Interactive html report

**customhtml**

Interactive html report containing custom charts

You can specify an output filename for each report.

**--target-fps=int**

The target frame rate in frames per second. Default = 60.

**-V, --version**

Print version information and exit.

Options for report metadata:

**--application-name=string**

The human readable name of the application being analyzed. For example, "Awesome Game". If the name contains whitespace, use quotes. This name becomes the report title. Default = "Performance Advisor Report".

**--build-name=string**

The build name of your application. For example, `nightly. fa34c92`.

**--build-timestamp=string**

The timestamp of your application build. For example, `Thu, 22 Aug 2019 12:47:30`.

**--device-name=string**

The name of the device that is used to obtain the capture.

**--package-name=string**

The name of the package used in the capture. If not provided, Performance Advisor tries to find the package name in the capture files. For example, `com.org.package`.

**--activity-name=string**

The name of the activity used from the package. If not provided, Performance Advisor tries to find the activity name in the capture files. For example, `.MainActivity`.

**--activity-args=string**

The arguments passed to the activity that ran during the capture. Surround the arguments with quotes if they contain whitespaces. If not provided, Performance Advisor tries to find the activity arguments in the capture files. For example, `-p --source directory`.

Options for setting a per-frame budget:

**--bandwidth-budget=<value>**

Threshold for read/write bytes.

**--cpu-cycles-budget=<value>**  
Threshold for CPU cycles.

**--draw-calls-budget=<value>**  
Threshold for draw calls.

**--gpu-cycles-budget=<value>**  
Threshold for GPU cycles.

**--overdraw-budget=<value>**  
Threshold for overdraw.

**--pixels-budget=<value>**  
Threshold for pixels.

**--primitives-budget=<value>**  
Threshold for primitives.

**--shader-cycles-budget=<value>**  
Threshold for shader cycles.

**--vertices-budget=<value>**  
Threshold for vertices.

Options for creating a custom chart:

**--custom-report=path**  
The path to the JSON report containing the custom chart definitions.

**--chart-list-output=path**  
Output location of the file containing chart names for the Streamline capture.

Options for creating a diff report:

**--diff-report-output=path**  
Output location for the diff report.

## Operation

You can pass options directly to the `streamline-cli -pa` command or you can list command-line options in a file that you pass to the `streamline-cli -pa` command.

If you pass the options as a command-line options file, specify one option per line and use `=` to assign values.

### Example: Passing options to `Streamline-cli -pa` using a command-line options file

For example, you might create a file for your budget thresholds called `budget` that contains the following options:

```
--build-name=8.2
--build-timestamp=3rd March 2021
--application-name=My Awesome Game
--cpu-cycles-budget=100000000
--gpu-cycles-budget=28000000
--shader-cycles-budget=20000000
```

```
--draw-calls-budget=350  
--vertices-budget=1000000
```

For options that accept a string, such as `--build-name`, `--build-timestamp`, or `--application-name`, note that the string does not need to be enclosed within quotes when it contains multiple words.

When you run Performance Advisor, specify the file with "`@<filename>`". In this example, you would use:

```
Streamline-cli -pa capture.apc "@budget"
```

## 6.2 The `streamline_me.py` script options

To see the possible options and their default values for the `streamline_me.py` command, run `python3 streamline_me.py -h`.

### Syntax

```
python3 streamline_me.py [OPTIONS]
```

### Options

**--device OR -E**

The target device name. The default is auto-detected.

**--package OR -P**

The application package name. The default is auto-detected.

**--package-activity**

The name of the application activity to start. If you do not specify this option, you must manually start the activity. This option is only supported for headless captures.

**--package-arguments**

The arguments to pass to the started application activity. This is only supported for headless captures.

**--headless OR -H**

Perform a headless capture, and write the result to a specified capture path. The default is to perform an interactive capture with no direct file output.

**--headless-timeout OR -T**

Exit the headless timeout after the specified number of seconds. The default is to wait for process exit.

**--config OR -C**

Specify the filename of the configuration XML file you want to use for a headless capture.

**--daemon OR -D**

Specify the path to the `gatord` binary you want to use if it is not found automatically. The default is auto-detected.

**--overwrite**

Overwrite an earlier headless output. The default is to error if the output file already exists.

**--verbose OR -v**

Enable verbose logging. The default is non-verbose logging.

**--lwi-mode OR -M**

Specify the mode of the interceptor that you want to use. The default is `off`.

The available values are:

- `off` if you do not want to use the light-weight interceptor, and use application-generated annotations instead.
- `counters` if you want to use the light-weight interceptor to generate frame boundaries and software counters.
- `screenshots` if you want to use the light-weight interceptor to generate frame boundaries, software counters, and slow frame screenshots.

**--lwi-api gles | vulkan**

Select the API you want to capture. The default is `gles`.

**--lwi-compress-img OR -X**

Enable compression of captured slow frame images, reducing file size but increasing the on-device CPU processing cost. The default is no compression.

**--lwi-gles-layer-name <name>**

The OpenGL ES layer name. The default is `libGLESLayerLWI.so`.

**--lwi-gles-layer-lib-path <path>**

The path to the OpenGL ES layer library. The default is auto-detected.

**--lwi-vk-layer-name <name>**

The Vulkan layer name. The default is `VK_LAYER_ARM_LWI`.

**--lwi-vk-layer-lib-path <path>**

The path to the Vulkan layer library. The default is auto-detected.

**--lwi-fps-window OR -W**

Specify the number of frames in the sliding window used for FPS calculation. The default is 6.

**--lwi-fps-threshold OR -Th**

Capture screenshots when the frame rate drops below this threshold. The default is 55.

**--lwi-frame-start OR -S**

Start tracking from a specified frame number. The default is frame 1.

**--lwi-frame-end OR -N**

End tracking at the specified frame number. The default is no end frame.

**--lwi-frame-gap Or -G**

Minimum number\_of\_frames between two slow frame captures. The default is 200 frames.

**--lwi-out-dir Or -o**

Specify the path to a directory for the captured images. This directory must be empty.