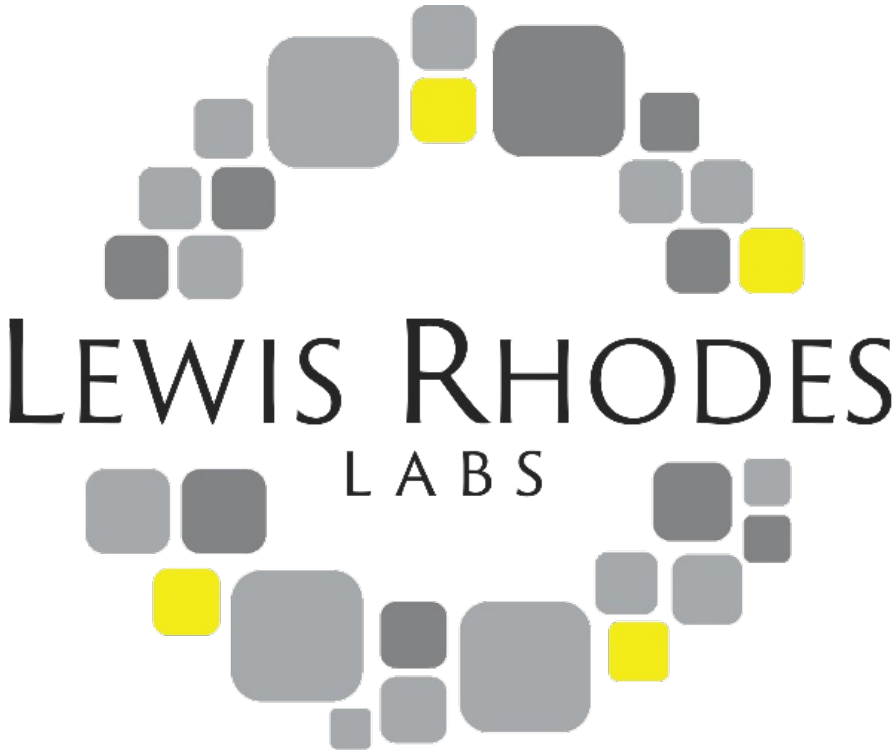

Extreme Search® Manual

Version 2024.10

https://lewis-rhodes.com

support@lewis-rhodes.com

---

1 Quick Start

1.1 Overview

Extreme Search® is a computational storage appliance that enables fast, fixed-throughput regular expression based search of files. The unique capability that Extreme Search® provides is that search time is independent of expression complexity or number. NPU*search* is a specific

implementation of a neuromorphic processor, invented by LRL, and optimized for search. NPU*search* IP allows all files on an appliance to be searched within 12 or 25 minutes (depending on model). Extreme Search® functionality is exposed through a Python library; users submit a list of regular expressions to search for and a list of files to search. Extreme Search® returns information about which files matched which expressions.

1.2 Adding data

Searches execute against stored data. The Extreme Search® appliance comes with the option to store data under GlusterFS, an open source distributed file system that scales to multiple petabytes of storage. A user interacts with Gluster in the same way as any normal file system. LRL preconfigures each appliance with a 1-node, multi-brick (exact number depends on model) Gluster volume named `gv0`. This volume is auto-mounted at `/mnt/gv0` at boot time. Gluster need not be used, however, to maximize the performance of Extreme Search® the data must be evenly distributed across the SSDs.

1.3 Extreme Search® software architecture

The search capabilities of Extreme Search® are enabled with 3 interacting software components:

1. **Backend**: Each backend is a NPU*search* processer which executes search on locally residing SSDs. Each Extreme Search® appliance runs multiple backends (exact number depends on model).
2. **Redis**: Redis mediates all communication between front end(s) and backends.
3. **Client Access (Frontend)**: The frontend exposes search functions, parses search requests, distributes work to backends, and returns results to user. Access is provided by the `npusearch` python package.

A software license is required for Extreme Search® to function. The license file should be in `/opt/lrl/lib/npusearch/`.

Details about where the redis server is, what the Gluster hosts are, where logs are stored, etc. can be configured in the backend configuration file located at `/opt/lrl/etc/npusearch.conf`.

1.4 "Hello World" example

Here's a simple search followed by a slightly more in-depth analysis on the matching file(s). You can copy/paste this directly into a `.py` file to run as a script. (*Note: This example assumes the gluster volume is named `demo` and is hosted on the ExtremeSearch® appliance itself. See the python documentation included with `npusearch` for more detail.*)

```
        #!/usr/bin/env python3

        import npusearch
        import os, time, pprint, re

        # Start by initializing the npu_client with info about the gluster configuration
        hosts = [os.uname().nodename]
        volname = 'demo'

        npu_client = npusearch.NPUGlusterClient(gluster_hosts=hosts, gluster_volname=volname)

        # This script uses wikipedia data that we have on many machines, but does not
        # come default.
        files = ['wiki/*.wiki'] # paths are relative to the gluster volume
```

```
        exprs = [r'(?i)celebratory.*neuroscience']

        # looking up this information directly can be slow with gluster,
        # so helper methods under the npu_client are provided
        file_sizes = npu_client.sizes(files)
        num_files = len(file_sizes['files'])

        totalbytes = file_sizes['size']
        totaldata = totalbytes/10**9 # convert to GB

        print(f'We will be searching {num_files:,} files containing '
              f'{totaldata/10**3:0.3f} TB of data.')
        print(f'Average file size: {totalbytes / num_files / 10**6 :0.0f} MB.')
        # We will be searching 23,032 files containing 2.474 TB of data.
        # Average file size: 107 MB.

        print(f'Running `npu_client.scan(exprs={exprs!r}, files={files!r})`.')
        t0 = time.perf_counter()
        npu_res = npu_client.scan(exprs=exprs, files=files) # main function
        t1 = time.perf_counter()

        npu_runtime = t1 - t0

        print(f'The NPU scanned {totaldata/10**3:0.3f} TB in {npu_runtime:0.2f} sec,'
              f' for a rate of {totaldata/npu_runtime:0.3f} GB/sec.')
        print(f'{len(npu_res["matches"])} out of {num_files:,} files matched.')
        # The NPU scanned 2.474 TB in 26.78 sec, for a rate of 92.376 GB/sec.
        # 1 out of 23,032 files matched.

        print('-'*30, '\nNPU output:')
        pprint.pprint(npu_res)
        # {'bytes_scanned': 2473900387015,
        #  'errors': [],
        #  'exceptions': {},
        #  'files_scanned': 23032,
        #  'matches': [{'brick': 'guava:/mnt/npusearch_5/demo/wiki/00000265.wiki',
        #               'matches': [0],
        #               'overflow': False,
        #               'path': 'wiki/00000265.wiki'}],
        #  'pe_usage': 8}

        # How to read the match results:
        # - `'matches': [0]` tells you that exprs[0] matched this file
        # - `'overflow': False` tells you that every expression in exprs which matched was
        #    reported (if more than 16 different expressions match, overflow may be True)
        # - `'pe_usage': 8` tells you how much space the compiled expressions took up on
        #    the NPU. Each scan can use up to 300 PEs (Processing Elements).

        with open(f"/mnt/{volname}/{npu_res['matches'][0]['path']}", 'rb') as f:
            re_res = re.search(exprs[0].encode('latin_1'), f.read())
        print('-'*30, '\nPython re result on matching file:')
        print(re_res[0])
        # b'celebratory responses, laughter, or [[self-serving bias]] in interpreting events.\
        # <ref name="SA_NTG">{{cite journal|title=The Neuroscience'
```

All NPU*search* specific capability is located in the `npusearch` module. See `Python Documentation` in the manual for the documentation. Alse, note that docstrings are included for each function, eg `help(npusearch.NPUGlusterClient.scan)`.

1.5 Supported (regular) expressions

Generally speaking, NPU*search* supports the Extended Network (EXNET) subset of regular expressions. Informally, an expression is NOT in EXNET if there is infinite repetition (`*`, `+`, or `{n,}`) on a pattern that is longer than a single character.

As an example, the expressions `abc(de)*fg`, `abc(d|e)+fg`, and `abc((d)?){5,}ef` are not supported,

while the expressions `abc[a-z0-9]*de`, `abc\w+de`, and `abc{3,}ef` are supported.

Extreme Search® supports assertions, line anchors, and look-behinds. Extreme Search® does not support look-aheads. Extreme Search® does not support back-references, recursion, or any other syntax that is outside the mathematical definition of a "regular expression".

For convenience, Extreme Search® supports an additional syntax for "at least *n*" (also known as "N of M") matching and logical combination matching. See section 2 of the `Expression Syntax` part of the manual for full details.

The complete syntax documentation can be found in the `Expression Syntax` part of the manual.

Since NPU*search* returns only whether or not the file matched an expression, match data such as subpattern matches or match location cannot be extracted from the results.

1.6 Frequently Asked Questions

1.6.1 Summary

Q1. What's a concise description of Extreme Search®'s functionality?

A1. Extreme Search® is a computational storage appliance that enables fast, fixed-throughput regular expression based search of files.

Q2. What existing search tool(s) is NPU*search* most similar to?

A2. Extreme Search® is similar to grep (https://linux.die.net/man/1/grep) or ripgrep (https://github.com/BurntSushi/ripgrep) in functionality, but faster and constant speed.

1.6.2 Functionality

Q1. My scan returns with `'pe_count': 0`. What happened?

A1. The backends may be powered off. Try running `client.check()` in your python. If an empty list is returned, then the backends are powered off. To power the backends back on, run `sudo systemctl restart npusearch.service`. To confirm that the backends are powered on, run `client.check()`, and confirm that the returned list has all of the backends. Alternatively, running `npusearch_check` from the command line returns the same result as `client.check()` in python.

Q2. My backends will not power on. What do I do?

A2. Confirm that your license is present and covers your current release version. To inspect your release version, inspect `/opt/lrl/VERSION` and see what `LICENSE_VERSION=` states. To inspect what versions your license covers, inspect your `.lic` file in `/opt/lrl/lib/npusearch/` and see what the date is. If your release version date is later then the date in the license file you need either a new license or to install an older version of Extreme Search®, to make your backends work.

If the license is correct, please see the `Common Issues` part of the manual. If that is not possible or the problem still persists, contact LRL at support@lewis-rhodes.com.

Q3. What does `scan()` return other than a list of matching files/expressions? Does it return number of matches, matching text, or match location?

A3. No; unlike some regular expression search tools, `scan()` is really simple - it tells you which files, in the list you pass it, matched which of the expressions you pass it. That's it. It doesn't tell you what the matching text was, where it matched, or how many matches there were. This is why we describe the typical use case as a "prefilter" - we reduce a customer's dataset to relevant files, and then tell them to run the analysis they were already going run (to find matching text, etc) on the reduced dataset. This is also why, earlier in this guide, we show a `scan()` search on the wiki data followed by a Python `re.search()` call on the single file that matches to get the matching text, offsets, etc. Replace `re.search()` with whatever tool or tools are relevant for your needs.

Q4. What factors influence the speed of `scan()`? If I pass in many expressions, will `scan()` run slower? Will `scan()` always take 12-25 minutes to run?

A4. The speed of `scan()` is asymptotically linear in file size and is independent of expression complexity or number as long as the expression fits on the device. By default, compiled expressions must fit within 300 PEs. Note in the example above that `'pe_usage': 8` indicates only 8 of the 300 PEs were used. The data passes through all 300 PEs, even if only 8 are used, so search time will not change depending on how many of the 300 PEs are used. Each backend searches the data which is local to its corresponding SSD(s) in asymptotically linear time corresponding to how much data is searched, and results are returned when each backend is finished. The speed of `scan()` is thus the speed of the slowest backend, so to minimize runtime the data must be as evenly distributed over the SSDs as possible. The 12-25 minute number is the time needed to search all of the data on all of the SSDs when all of the SSDs are complelety full of data; searching a small fraction of the data can be as fast as 200 milliseconds.

Passing `decompression=True` (or `'auto'`) to `scan()` will cause files to pass through the CPU before passing through the NPU, which will hurt the performance of `scan()`.

Q5. Does NPU*search* support compression/decompression?

A5. As of this release we now support decompression. See the `Architecture` section of the manual for details.

Q6. Does NPU*search* support encryption/decryption?

A6. No, it currently does not. That being said, Samsung's SmartSSDs, which are included in one of the Extreme Search® models, support self-encrypting drive (SED) functionality that can be transparently enabled so all data at rest is encrypted. Kuona models can come with SED functionality as well, please make sure you indicate you want this when placing any orders.

Q7. Is NPU*search* applicable to field x?

A7. NPU*search* is applicable to any field or any dataset where the granularity of the data is the file, and where the data within the file is amenable to regular expression based search. In the field of cyber forensics, we expect NPU*search* to excel at searching plaintext log files (like Bro/Zeek logs). It may also be adequate at searching PCAP network captures, though with the caveat that it doesn't do packet parsing, TCP stream reassembly, or gzip decompression. In the case where images are a primary form of data, NPU*search* may be excellent at searching the metadata associated with each image.

1.7 Support

Please submit questions, comments, and/or issues to LRL at support@lewis-rhodes.com.