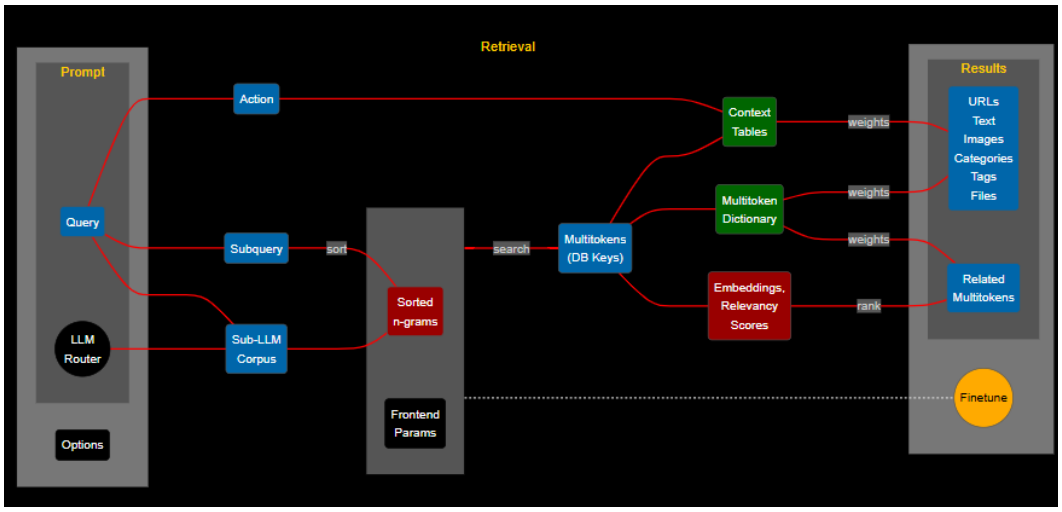
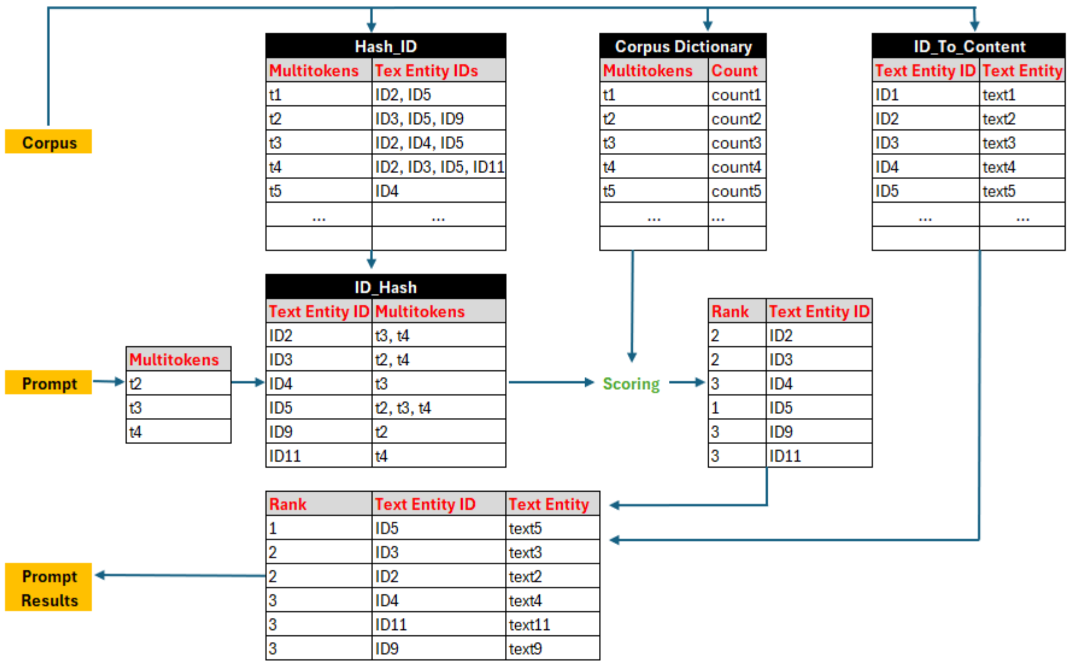
Building Disruptive AI & LLM Technology From Scratch





Hash_ID

Multitokens	Tex Entity IDs
t1	ID2, ID5
t2	ID3, ID5, ID9
t3	ID2, ID4, ID5
t4	ID2, ID3, ID5, ID11
t5	ID4
...	...

Corpus Dictionary

Multitokens	Count
t1	count1
t2	count2
t3	count3
t4	count4
t5	count5
...	...

ID_To_Content

Text Entity ID	Text Entity
ID1	text1
ID2	text2
ID3	text3
ID4	text4
ID5	text5
...	...

Multitokens
t2
t3
t4

ID_Hash

Text Entity ID	Multitokens
ID2	t3, t4
ID3	t2, t4
ID4	t3
ID5	t2, t3, t4
ID9	t2
ID11	t4

Rank	Text Entity ID
2	ID2
2	ID3
3	ID4
1	ID5
3	ID9
3	ID11

Rank	Text Entity ID	Text Entity
1	ID5	text5
2	ID3	text3
2	ID2	text2
3	ID4	text4
3	ID11	text11
3	ID9	text9

Vincent Granville, Ph.D. | www.MLTechniques.com | Version 3.0, November 2024

Contents

Introduction

This book features new advances in game-changing AI and LLM technologies built by GenAItechLab.com. Written in simple English, it is best suited for engineers, developers, data scientists, analysts, consultants and anyone with an analytic background interested in starting a career in AI. The emphasis is on scalable enterprise solutions, easy to implement, yet outperforming vendors both in terms of speed and quality, by several orders of magnitude.

Each topic comes with GitHub links, full Python code, datasets, illustrations, and real life case studies, including from Fortune 100 company. Some of the material is presented as enterprise projects with solution, to help you build robust applications and boost your career. You don't need expensive GPU and cloud bandwidth to implement them: a standard laptop works.

Part I: Hallucination-Free LLM with Real-Time Fine-Tuning focuses on high performance in-memory agentic multi-LLMs for professional users and enterprise, with real-time fine-tuning, self-tuning, no weight, no training, no latency, no hallucinations, no GPU. Made from scratch, leading to replicable results, leveraging explainable AI, adopted by Fortune 100. With a focus on delivering concise, exhaustive, relevant, and in-depth search results, references, and links. See also the section on 31 features to substantially boost RAG/LLM performance.

Part II: Outperforming Neural Nets and Classic AI discusses related large-scale systems also benefiting from a light-weight but more efficient architecture. It features LLMs for clustering, classification, and taxonomy creation, leveraging the knowledge graphs embedded in and retrieved from the input corpus when crawling. Then, in chapters 7 and 8, I focus on tabular data synthetization, presenting techniques such as NoGAN, that significantly outperform neural networks, along with the best evaluation metric. The methodology in chapter 9 applies to most AI problems. It offers a generic tool to improve any existing architecture relying on gradient descent, such as deep neural networks. Finally, chapter 10 discusses deep retrieval and multi-index chunking for PDF repositories, capturing elements missed by most RAG/LLM systems, with Nvidia case study.

Part III: Innovations in Statistical AI features a collection of methods that you can integrate in any AI system to boost performance. Based on a modern approach to statistical AI, they cover probabilistic vector search, sampling outside the observation range, strong random number generators, math-free gradient descent, beating the slow statistical convergence of parameter estimates dictated by the Central Limit Theorem, exact geospatial interpolation for non-smooth systems, and more. Efficient chunking and indexing for LLMs is the topic of chapter 11. Finally, chapter 16 shows how to optimize trading strategies to consistently outperform the stock market.

About the author

Vincent Granville is a pioneering GenAI scientist and machine learning expert, co-founder of Data Science Central (acquired by a publicly traded company in 2020), Chief AI Scientist at MLTechniques.com, former VC-funded executive, author and patent owner – one related to LLM. Vincent's past corporate experience includes Visa, Wells Fargo, eBay, NBC, Microsoft, and CNET.

Vincent is also a former post-doc at Cambridge University, and the National Institute of Statistical Sciences (NISS). He published in *Journal of Number Theory*, *Journal of the Royal Statistical Society* (Series B), and *IEEE Transactions on Pattern Analysis and Machine Intelligence*. He is the author of multiple books, available here, including "Synthetic Data and Generative AI" (Elsevier, 2024). Vincent lives in Washington state, and enjoys doing research on stochastic processes, dynamical systems, experimental math and probabilistic number theory.

Chapter 10

Deep Retrieval and Multi-Index Chunking for PDFs

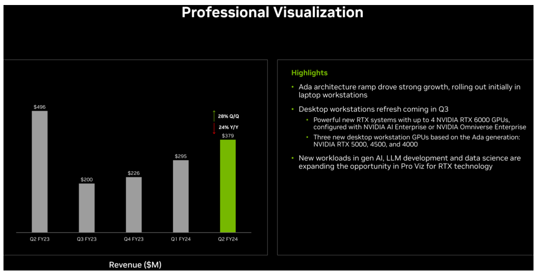
In this chapter, I discuss the preprocessing steps used to turn a PDF repository into input suitable for xLLM. It includes chunking, indexing text entities with hierarchical multi-index system, retrieving contextual elements including font type, color, and size, building additional contextual information such as agents to add to text entities, as well as retrieving images and tables – some not detected by standard Python libraries, relying instead on proprietary technology.

Section 10.2 includes a discussion on why training an LLM for token prediction is not needed, how you can do without to save considerable training time and minimize costs (no GPU, zero weight), as well as current challenges with model evaluation. Section 10.3 summarizes key features of xLLM, contrasting them with other systems, and justifying the label LLM 2.0 for our innovative, radically different architecture with its numerous next-gen subsystems.

10.1 PDF contextual parsing and chunking: Nvidia case study

Here, I illustrate how you can perform chunking, multi-indexing, table and image retrieval on a PDF repository featuring public financial reports from Nvidia. The goal is to turn the PDFs into a format amenable to xLLM, with contextual chunks similar – at least from a design standpoint – to the one featured in Table 2.1. Although the Python code handles multiple PDFs at once, I focus on a single PDF document, posted here.

I use the PyMuPDF Python library also known as Fitz, with home-made algorithms to retrieve tables that it fails to detect. Another useful library is LlamaParse. Yet, a different approach consists of saving each PDF page (a slide in this case) as an image, then using computer vision technology to retrieve the various elements from the images, and turn them into text when appropriate. Indeed, lines 286–300 in the code in section 10.1.3 turn the slides into images. However, I will not follow this approach. Instead, I convert the PDFs to JSON (line 158 in the code) and then parse the JSON elements, including images. Note that it includes non-standard characters: see how to process them in lines 85-91 in the code.



Professional Visualization

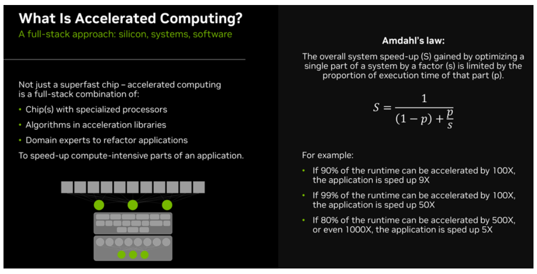
Revenue ($M)

	Q2 FY23	Q3 FY23	Q4 FY23	Q1 FY24	Q2 FY24
	$496	$200	$226	$295	$379

28% Q/Q
24% Y/Y

Highlights

- Ada architecture ramp drove strong growth, rolling out initially in laptop workstations
- Desktop workstations refresh coming in Q3
  - Powerful new RTX systems with up to 4 NVIDIA RTX 6000 GPUs, configured with NVIDIA AI Enterprise or NVIDIA Omniverse Enterprise
  - Three new desktop workstation GPUs based on the Ada generation: NVIDIA RTX 5000, 4500, and 4000
- New workloads in gen AI, LLM development and data science are expanding the opportunity in Pro Viz for RTX technology



What Is Accelerated Computing?

A full-stack approach: silicon, systems, software

Not just a superfast chip – accelerated computing is a full-stack combination of:

- Chip(s) with specialized processors
- Algorithms in acceleration libraries
- Domain experts to refactor applications

To speed-up compute-intensive parts of an application.

Amdahl's law:

The overall system speed-up (S) gained by optimizing a single part of a system by a factor (s) is limited by the proportion of execution time of that part (p).

$$S = \frac{1}{(1-p) + \frac{p}{s}}$$

For example:

- If 90% of the runtime can be accelerated by 100X, the application is sped up 9X
- If 99% of the runtime can be accelerated by 100X, the application is sped up 50X
- If 80% of the runtime can be accelerated by 500X, or even 1000X, the application is sped up 5X

Figure 10.1: Nvidia PDF: slide with sublist, hidden table (left); with 2 lists and formula (right)

Two slides (thumbnails) are featured in Figure 10.1. For full resolution, click on the corresponding thumbnail, or zoom in. The resulting text entities are shown respectively in Tables 10.1 and 10.2, with entries (rows) listed in the order they were retrieved: sometimes, the slide title is not the first element to be recovered, and sometimes the title is absent. The remaining of the discussion focuses on how to build these text entities, also adding multimodal elements such as tables and images, and how to integrate them with xLLM.

Table 10.1: Turning the left slide in Figure 10.1 into a multi-index xLLM text entity

Type	ID1	ID2	ID3	Size	Text
Note	0	-1	-1	36	|Highlights|
List	0	0	-1	36	Growth fueled by GeForce RTX 40 series GPUs for laptops and desktops
List	0	1	-1	36	Large upgrade opportunity ahead: just 47% of installed base have upgraded to RTX; ∼20% have a GPU with RTX 3060 or higher
List	0	2	-1	36	NVIDIA GPU-powered laptops have gained in popularity; shipments now outpace desktop GPUs in several regions around the world
SubList	0	2	0	32	Likely to shift overall Gaming seasonality a bit, with Q2 and Q3 stronger, reflecting the back-to-school and holiday build schedules
List	0	3	0	36	RTX/DLSS ecosystem continues to expand; 35 new games added DLSS support, including Diablo IV and Baldur's Gate 3; there are now over 330 RTX accelerated games and apps
List	0	4	0	36	Bringing generative AI to gaming with NVIDIA Avatar Cloud Engine (ACE) for Games
Title	1	4	0	72	Gaming
Note	3	-1	-1	40	|Revenue ($M)|
Data	4	TD0	-1	24	|$2,042|$1,574|$1,831|$2,240|$2,486|
Note	5	TL0	-1	24	|Q2 FY23|Q3 FY23|Q4 FY23|Q1 FY24|Q2 FY24|
Data	6	-1	-1	24	|22% Y/Y|
Note	7	-1	-1	24	|and|
Data	8	-1	-1	24	|11% Q/Q|

Table 10.2: Turning the right slide in Figure 10.1 into a multi-index xLLM text entity

Type	ID1	ID2	ID3	Size	Text
Title	0	-1	-1	72	What Is Accelerated Computing?
Note	1	-1	-1	44	|Not just a superfast chip – accelerated computing|is a full-stack combination of:|
List	1	0	-1	44	Chip(s) with specialized processors
List	1	1	-1	44	Algorithms in acceleration libraries
List	1	2	-1	44	Domain experts to refactor applications To speed-up compute-intensive parts of an application.
Note	3	-1	-1	48	|A full-stack approach: silicon, systems, software|
Note	4	-1	-1	44	|For example:|
List	4	0	-1	44	If 90% of the runtime can be accelerated by 100X, the application is sped up 9X
List	4	1	-1	44	If 99% of the runtime can be accelerated by 100X, the application is sped up 50X
List	4	2	-1	44	If 80% of the runtime can be accelerated by 500X, or even 1000X, the application is sped up 5X
Note	6	-1	-1	48	|Amdahl's law:|
Note	7	-1	-1	44	|The overall system speed-up (S) gained by optimizing a|single part of a system by a factor (s) is limited by the|proportion of execution time of that part (p).|
Note	8	-1	-1	60	|\U0001d446=|
Data	9	TD0	-1	60	|1|1\u2212 \U0001d45d+|
Note	10	TL0	-1	60	|\U0001d45d|\U0001d460|

10.1.1 Multi-index, bullet lists, images, tables

A multi-index is a vector index, in contrast to univariate indexes discussed in chapter 11 and found in standard LLMs. In our example, the columns ID1, ID2, ID3 and Size in Tables 10.1 and 10.2 are part of the multi-index. In the Python code, they corresponds respectively to `block_ID`, `item_ID`, `sub_ID`, and `font_size`: see code line 240. Other multi-index components generated by the Python script in the same code line: `doc_ID` (reference to the parent PDF document), `page_num` (the page number associated to the text entity in the PDF document in question), `font_type`, `font_color` and `block_number`.

10.1.1.1 Multi-index components

I now describe the main components ID1, ID2, ID3 and so forth built into the multi-index.

- ID1 (`block_ID`) is an home-made alternate block identifier to the JSON `block_number` generated by the PyMuPDF library. It puts together broken sentences having multiple block numbers in JSON. Typically, the text sections separated by "|" in the Text column in Tables 10.1 and 10.2 are assigned different block numbers, yet are considered to be part of a same `block_ID`.
- ID2 (`item_ID`) and ID3 (`sub_ID`): see section 10.1.1.5
- Size (`font_size`) identifies the size of the font, an indicator of importance. Also, color and font type (for instance, italics or bold) convey special information. The Python code retrieves these elements.
- In addition, `doc_ID` is used to identify the parent document in a repository with multiple PDFs, while `page_num` identifies the page number in a same PDF, whether pages are numbered or not.

10.1.1.2 Images

Images also have their own multi-index, consisting of `doc_ID`, `page_num`, and the image number in the text entity, represented by `image_index` (line 261 in the code). To save these images to output files, uncomment lines 263–264 in the code. These images are internal to the PDFs and should not be confused with images created for each slide in lines 288–300. Typically, they represent logos, part of a diagram, or full-fledged valuable images. By looking at the size and type distribution (lines 259–260), it is easy to detect and discard useless images that appear on many slides. Indexed images (type + size) are listed in the output file, at the bottom of each text entity: see here.

10.1.1.3 Tables

The tables detected by the PyMuPDF library in lines 160–170 are not indexed yet. However, those internally detected have a special ID2 in the multi-index, consisting of the letter T (for table), followed by the letter D for data or L for labels, and then followed by the table number (`table_ID`, line 52) in the text entity in question. See example in Table 10.1.

10.1.1.4 Text entities, sub-entities

I use the words chunk and text entity interchangeably, with the same meaning throughout this book. Note that in many standard LLMs, chunks are not determined by the document structure (they are fixed-size) and do not include contextual information. This is not the case with xLLM. Finally, a sub-entity is one element in a text entity: in our example, one row in Tables 10.1 or 10.2. More on text entities in section 10.1.2.

10.1.1.5 Bullet lists

Bullet lists are not recognized by the PyMuPDF library, yet are everywhere. A bullet list is identified by three IDs, namely ID1, ID2, and ID3. All items within a same list (including sub-lists) have the same ID1, representing the list number within a same text entity, itself identified by `page_num`. For instance, we have two distinct bullet lists in Table 10.2, and one list with a sub-list in Table 10.1. Sub-list items have an ID3 distinct from -1, and the same ID2 identifying the parent list item. Different list items at the top level have different ID2's but the same ID1.

10.1.2 Diagrams, context, agents, auto-tagging

Each text entity is assigned a type: data, labels, title, note, list, or sub-list. In particular, "title" is a contextual sub-entity, as opposed to a regular one. In xLLM, multi-tokens found in contextual elements are called graph multitokens as they are related to the underlying knowledge graph, and are treated separately: see section 11.2. In the Nvidia corpus, another type of contextual information are keywords in green font. These are index keywords and play the same role as the words highlighted in orange in this book (all of them are index terms). Additional contextual sub-entities will be added later, including agents, tags, and categories, using an auto-tagging algorithm without human interaction. The first step consists of creating a multi-token dictionary and clustering techniques along with analyzing the token distribution, to identify candidate tags, categories, and agents. The process is similar to the one already in place for the corporate corpus discussed in section 2.3.

As for diagrams, they are detected as images and available in the individual slides stored as images. In many cases (histograms), the underlying table is retrieved using the mechanism described in section 10.1.1.3. It allows for automated predictive analytics on multiple tables blended together. Some diagrams can be rendered as HTML or JSON, as the PyMuPDF library allows you to convert PDF to JSON or HTML.

Finally, the PDFs contain many special characters, including those found in mathematical formulas. For instance, the bullet symbol indicating a bullet list, or the bottom three rows in Table 10.2. In this case, the special characters are converted to a code number, starting with \u or \U. See lines 85–91 in the Python code. Interestingly, line 86 in the Python code contains the symbol • without causing problems when executing it.

10.1.3 Python code

The Python code, input PDF and output (indexed contextual text entities in text format) are on GitHub, respectively here (code), here (input), and here (output).

```
1  # Input PDF for this script: https://drive.google.com/file/d/1Daa9oZJm4-b6NqUsVGxK2euemcFnf8jH/
2  
3  # https://pymupdf.readthedocs.io/en/latest/page.html#Page.find_tables
4  #
       https://stackoverflow.com/questions/56155676/how-do-i-extract-a-table-from-a-pdf-file-using-pymupdf
5  
6  import fitz # PyMuPDF
7  
8  
9  def update_item_ID(k, entity_idx, type, table_ID):
10 
11    idx = entity_idx[k]
12    if type == 'Data':
13       # table: data row
14       flag = 'TD' #
15    elif type == 'Note':
16       # table: labels
17       flag = 'TL'
18    idx_list = list(idx)
19    idx_list[1] = flag + str(table_ID)
20    entity_idx[k] = tuple(idx_list)
21 
22    return(entity_idx)
23 
24 
25 def detect_table(xLLM_entity):
26 
27    # detect and flag simple pseudo-tables
28 
29    entity_txt = xLLM_entity[0]
30    entity_type = xLLM_entity[1]
31    entity_idx = xLLM_entity[2]
32    table_ID = -1
33    table_flag = False
34 
35    for k in range(1, len(entity_type)):
36 
37       type = entity_type[k]
38       text = entity_txt[k]
39       old_text = entity_txt[k-1]
40       old_type = entity_type[k-1]
41 
42       if ( (
43            (type == 'Data' and old_type == 'Note') or
44            (type == 'Note' and old_type == 'Data') or
45            (type == 'Data' and old_type == 'Data')
46           )
47           and old_text.count('|') == text.count('|')
48           and text.count('|') > 2
49         ):
50          print("detected table", table_ID + 1)
51          if not table_flag:
```

```
            table_ID += 1
            table_flag = True
         idx = entity_idx[k]
         old_idx = entity_idx[k-1]

         # update item_ID (idx[1] and old_idx[1]) in current and previous row
         # item_ID starts with letter D if data, or N if labels

         update_item_ID(k, entity_idx, type, table_ID)
         update_item_ID(k-1, entity_idx, old_type, table_ID)

      else:
         #table_ID = -1
         table_flag = False

   return(xLLM_entity)


def cprint_page(xLLM_entity, OUT):

   entity_txt = xLLM_entity[0]
   entity_type = xLLM_entity[1]
   entity_idx = xLLM_entity[2]

   for k in range(len(entity_type)):

      type = entity_type[k]
      text = entity_txt[k]
      text = text.strip()
      text = text.replace("  ", " ")
      text = text.replace(" |", "|")
      text = text.replace("| ", "|")
      text = text.replace("||","|")
      text = text.encode('unicode-escape').decode('ascii')
      text = text.replace('\\u2022', '•')
      text = text.replace('\\u2013', '--')
      text = text.replace('\\u2014', '--')
      text = text.replace('\\u2019', "'")
      text = text.replace('\\u201c', '"')
      text = text.replace('\\u201d', '"')

      idx = entity_idx[k]
      doc_ID = idx[3]
      block_ID = idx[0]
      item_ID = idx[1]
      sub_ID = idx[2]
      pn = idx[4] # page number
      fs = idx[5] # font size
      fc = idx[6] # font color
      ft = idx[7] # font typeface
      #- print(k, type, idx, text)
      OUT.write(f"{type:<8}{block_ID:>3}{item_ID:>5}{sub_ID:>3}{pn:>3}"
              f"{fs:>5}{fc:>9} {ft:<20}{text:<80}\n")

   OUT.write("\n")
   return()


def update_page(text, type, entity, idx):

   entity_txt = entity[0]
   entity_type = entity[1]
   entity_idx = entity[2]
   block_ID = idx[0]
   item_ID = idx[1]
   sub_ID = idx[2]
   k = len(entity_txt)
   if k > 0:
      old_type = entity_type[k-1]
      old_idx = entity_idx[k-1]
      old_block_ID = old_idx[0]
      old_item_ID = old_idx[1]
      old_sub_ID = old_idx[2]
   else:
      old_type = ""
      old_block_ID = ""
      old_item_ID = ""
      old_sub_ID = ""
   if type in (’Note’, ’Data’):
      sep = "|"
   else:
      sep = " "

   if (type == old_type and block_ID == old_block_ID
        and item_ID == old_item_ID and sub_ID == old_sub_ID):
     new_text = entity_txt[k-1] + text + sep
     entity_txt[k-1] = new_text
   else:
      entity_txt.append(sep + text + sep)
      entity_type.append(type)
      entity_idx.append(idx)
   return(entity)


def convert_pdf_to_json(pdf_path, json_path, doc_ID, text_path):
   # Open the PDF file
   pdf_document = fitz.open(pdf_path)
   content = ""

   # Iterate through the pages
   for page_num in range(len(pdf_document)):
      OUT.write("\n----------------------------\n")
      OUT.write("Processing page " + str(page_num) + "\n\n")
      print("Page:", page_num)
      page = pdf_document.load_page(page_num)

      text_data = page.get_text("dict") # also extract as "json" to get tokens in green font

      tabs = page.find_tables()
      for tabs_index, tab in enumerate(tabs):
         # iterate over all tables
         index = (page_num, tabs_index)
         table_data = tab.extract() # extracting tabs[i], the i-th table in this page
         if len(table_data) > 0:
            # if not, ignore this table (note the important parameter threshold here)
            OUT.write("Table " + str(index) + ":\n")
            for row in table_data:
               OUT.write(str(row) + "\n")
            OUT.write("\n")

      itemize = False
      item_ID = -1
      sub_ID = -1
      block_ID = -1
      item = ""
      fsm = -1 # top level font size in bullet list
      fst = 64 # min title font size (top parameter)
      title = ""
      notes = ""
      old_block_number = -1
      old_font_size = -1
      entity_txt = []
      entity_idx = []
      entity_type = []
      type = ""
      old_text = ""
      old_type = ""

      for block in text_data["blocks"]:
         if block["type"] == 0: # Text block
            block_number = block["number"]
            for line in block["lines"]:
               for span in line["spans"]:

                  text = span["text"]
                  font_name = span["font"]
                  font_size = span["size"]
                  font_size = round(font_size,1)
                  font_color = span["color"]

                  if font_size > fst:
                     type = ’Title’
                  elif ord(text[0]) == 8226:
                     itemize = True
                     if fsm == -1:
                        fsm = font_size
                     if font_size > 0.98 * fsm:
                        item_ID += 1
                        type = 'List'
                     else:
                        sub_ID +=1
                        type = 'SubList'
                  elif itemize:
                     #- itemize = ((0.99 < font_size/old_font_size < 1.01) and
                     #-          (not text[0].isupper() or ord(old_text[0]) == 8226))
                     itemize = ((0.99 < font_size/old_font_size < 1.01) or
                             (ord(old_text[0]) == 8226))

                     if not itemize:
                        item_ID = -1
                        sub_ID = -1
                        block_ID += 1
                        type = 'Note'
                  else:
                     if not text[0].isdigit() and text[0] not in ('$', '+', '-'):
                        type = 'Note'
                     #- elif block_number != old_block_number:
                     else:
                        type = 'Data'

                  if block_ID == -1:
                     block_ID += 1
                  elif ((type not in (old_type, 'List', 'SubList')) or
                       (not (0.99 < font_size/old_font_size < 1.01))):
                     if (old_text != "" and ord(old_text[0]) != 8226 and
                           type not in ('List', 'SubList')):
                        block_ID += 1

                  idx = (block_ID, item_ID, sub_ID, doc_ID, page_num, font_size,
                       font_color, font_name, block_number)
                  entity = (entity_txt, entity_type, entity_idx)
                  update_page(text, type, entity, idx)

                  old_font_size = font_size
                  old_text = text
                  old_type = type

               old_block_number = block_number

      entity = detect_table(entity)
      cprint_page(entity, OUT)

      image_list = page.get_images()
      for image_index, img in enumerate(image_list, start=1):
         xref = img[0]
         base_image = pdf_document.extract_image(xref)
         image_bytes = base_image["image"]
         size = len(image_bytes)
         ext = base_image["ext"]
         index = (page_num, image_index)
         OUT.write(f"Image {str(index):<8}{str(ext):>5} size = {str(size):>6}\n")
         #- with open(f"image_{page_num + 1}_{image_index}.{ext}", "wb") as image_file:
         #-   image_file.write(image_bytes)

      content += "__P" + str(page_num) + "\n" + page.get_text("json") # "text", "html", "json"

   # Write the json content to a file
   with open(json_path, 'w', encoding='utf-8') as json_file:
      json_file.write(content)


# --- Main ---

doc_ID = 0 # to identify the PDF doc
filename = 'nvda-f2q24-investor-presentation-final-1'
pdf_path = filename + '.pdf'
json_path = filename + '.json'
text_path = filename + '.txt'
```

```python

OUT = open(text_path, "wt", encoding="utf-8")
convert_pdf_to_json(pdf_path, json_path, doc_ID, OUT)
OUT.close()


# --- PDF to Images ---

from PIL import Image

pdf_document = fitz.open(pdf_path)
zoom = 2 # to increase the resolution
mat = fitz.Matrix(zoom, zoom)

for page_num in range(len(pdf_document)):
   page = pdf_document.load_page(page_num)
   pix = page.get_pixmap(matrix = mat) # or (dpi = 300)
   img = Image.frombytes("RGB", [pix.width, pix.height], pix.samples)
   filename = "PDF" + str(page_num) + '.png' # you could change image format accordingly
   img.save(filename)
   print('Converting PDFs to Image ... ' + filename)
```

10.2 Challenges with standard LLM implementations

LLMs are trained on tasks irrelevant to what they will do for the user. It's like training a plane to efficiently operate on the runway, but not to fly. In short, it is almost impossible to train an LLM, and evaluating is just as challenging. Then, training is not even necessary. In this article, I dive on all these topics.

10.2.1 Training LLMs for the wrong tasks

Since the beginnings with Bert, training an LLM typically consists of predicting the next tokens in a sentence, or removing some tokens and then have your algorithm fill the blanks. You optimize the underlying deep neural networks to perform these supervised learning tasks as well as possible. Typically, it involves growing the list of tokens in the training set to billions or trillions, increasing the cost and time to train. However, recently, there is a tendency to work with smaller datasets, by distilling the input sources and token lists. After all, out of one trillion tokens, 99% are noise and do not contribute to improving the results for the end-user; they may even contribute to hallucinations. Keep in mind that human beings have a vocabulary of about 30,000 keywords, and that the number of potential standardized prompts on a specialized corpus (and thus the number of potential answers) is less than a million.

In addition, training relies on minimizing a loss function that is just a proxy to the model evaluation function. So, you don't even truly optimize next token prediction, itself a task unrelated to what LLMs must perform for the user. To directly optimize the evaluation metric, see my approach in chapter 9. And while I don't design LLMs for next token prediction, see one exception in section 7.1 in [13], to synthesize DNA sequences.

The fact is that LLM optimization is an unsupervised machine learning problem, thus not really amenable to training. I compare it to clustering, in contrast to to supervised classification. There is no perfect answer except for trivial situations. In the context of LLMs, laymen may like OpenAI better than my own technology, while the converse is true for busy business professionals and advanced users. Also, new keywords keep coming regularly. Acronyms and synonyms that map to keywords in the training set yet absent in the corpus, are usually ignored. All this further complicates training.

10.2.2 Analogy to clustering algorithms

As an example, this time in the context of clustering, see Figure 10.2. It features a dataset with three clusters. How many do you see, if all the dots had the same color? Ask someone else, and you may get a different answer. What characterizes each of these clusters? You cannot train an algorithm to correctly identify all cluster structures, though you could for very specific cases like this one. The same is true for LLMs. Interestingly, some vendors claim that their LLM can solve clustering problems. I would be happy to see what they say about the dataset in Figure 10.2.

For those interested, the dots in Figure 1 represent points on three different orbits of the Riemann zeta function. The blue and red ones are related to the critical band. The red dots correspond to the critical line with infinitely many roots. It has no hole. The yellow dots are on an orbit outside the critical band; the orbit in question is bounded (unlike the other ones) and has a hole in the middle of the picture. The blue orbit also appears to not cover the origin where red dots are concentrated; its hole center — the "eye" as in the eye of a hurricane — is on the left side, to the right of the dense red cluster but left to the center of the hole in the yellow cloud. Proving that its hole encompasses the origin, amounts to proving the Rieman Hypothesis. The blue and red orbits are unbounded.

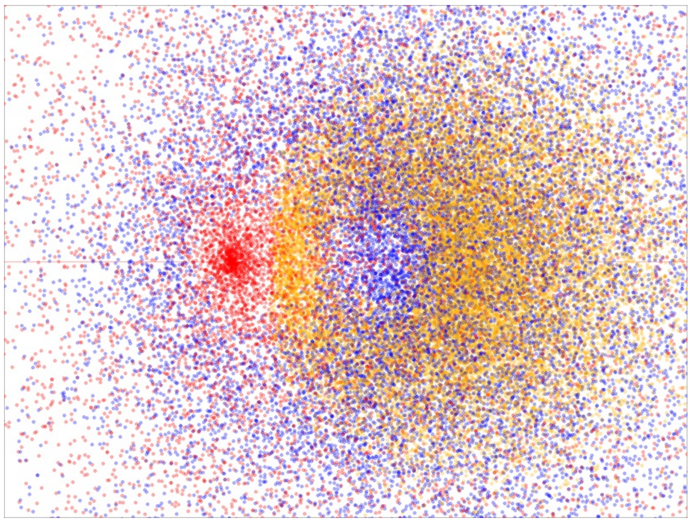


Figure 10.2: Training an LLM is as hard as training a clustering algorithm

Can LLMs figure this out? Can you train them to answer such questions? The answer is no. They could succeed if appropriately trained on this very type of example but fail on a dataset with a different structure (the equivalent of a different kind of prompt). One reason why it makes sense to build specialized LLMs rather than generic ones.

10.2.3 Challenges in evaluating LLMs

There are numerous evaluation metrics to assess the quality of an LLM, each measuring a specific type of performance. It is reminiscent of test batteries for random number generators (PRNGs) to assess how random the generated numbers are. But there is a major difference. In the case of PRNGs, the target output has a known, prespecified uniform distribution. The reason to use so many tests is because no one has yet implemented a universal test, based for instance on the full multivariate Kolmogorov-Smirnov distance. Actually, I did recently and will publish an article about it. It even has its Python library, here.

For LLMs that answer arbitrary prompts and output English sentences, there is no possible universal test. Even though LLMs are trained to correctly replicate the full multivariate token distribution — a problem similar to PRNGs thus with a universal evaluation metric — the goal is not token prediction, and thus evaluation criteria must be different. There will never be a universal evaluation metric except for very peculiar cases, such as LLMs for predictive analytics. Below is a list of important features lacking in current evaluation and benchmarking metrics.

10.2.3.1 Overlooked criteria, hard to measure

Some of the important features rarely present in LLM benchmarks include:

- Exhaustivity: Ability to retrieve absolutely every relevant item, given the limited input corpus. Full exhaustivity is the ability to retrieve everything on a specific topic, like chemistry, even if not in the corpus. This may involve crawling in real-time to answer the prompt.
- Inference: Ability to invent correct answers for problems with no known solution, for instance proving or creating a theorem. I asked GPT if the sequence (a^n) is equidistributed modulo 1 if $a = 3/2$. Instead of providing references as I had hoped – it usually does not – GPT created a short proof on the fly, and not just for $a = 3/2$; Perplexity failed at that. The use of synonyms and links between knowledge elements can help achieve this goal.
- Depth, conciseness and disambiguation are not easy to measure. Some users see long English sentences superior to concise bullet lists grouped into sections in the prompt results. For others, the opposite is true. Experts prefer depth, but it can be overwhelming to the layman. Disambiguation can be achieved by asking the user to choose between different contexts; this option is not available in most LLMs. As a

result, these qualities are rarely tested. The best LLMs, in my opinion, combine depth and conciseness, providing in much less text far more information than their competitors. Metrics based on entropy could help in this context.

- Ease of use is rarely tested, as all UIs consist of a simple search box. However, xLLM is different, allowing the user to select agents, negative keywords, and sub-LLMs (subcategories). It also allows you to fine-tune or debug in real-time thanks to intuitive parameters. In short, it is more user-friendly. For details, see here.
- Security is also difficult to measure. But LLMs with local implementation, not relying on external APIs and libraries, are typically more secure. Finally, a good evaluation metric should take into account the relevancy scores displayed in the results. As of now, only xLLM displays such scores, telling the user how confident it is in its answer.

10.2.4 Benefits of untrained LLMs

In view of the preceding arguments, one might wonder if training is necessary, especially since training is not done to provide better answers, but to better predict the next tokens. Without training, its heavy computations and black-box deep neural networks machinery, LLMs can be far more efficient yet even more accurate. To generate answers in English prose based on the retrieved material, one may use template, pre-made answers where keywords can be plugged in. This can turn specialized sub-LLMs into giant Q&A's with a million questions and answers covering all potential inquiries. And deliver real ROI to the client by not requiring GPU and expensive training.

This is why I created xLLM, and the reason why I call it LLM 2.0, with its radically different architecture and next-gen features including the unique UI. I describe it in detail in Part I in this book. Additional research papers are available here.

That said, xLLM still requires fine-tuning, for back-end and front-end parameters. It also allows the user to test and select his favorite parameters. Then, it blends the most popular ones to automatically create default parameter sets. I call it self-tuning. It is possibly the simplest reinforcement learning strategy.

10.3 Overview of xLLM main differentiators

Due to the innovative architecture and next gen features, xLLM constitutes a milestone in LLM development, moving away from the deep neural network (DNN) machinery and its expensive black-box training and GPU reliance, while delivering more accurate results to professional users, especially for enterprise applications. In Table 10.3, I summarize some of the most innovative contributions, justifying the label LLM 2.0. In the table below, KG stands for knowledge graph.

Table 10.3: Comparing LLM 2.0 with LLM 1.0

xLLM (LLM 2.0)	Standard LLMs (LLM 1.0)
Solid foundations to design robust back-end architecture from the ground up, retrieve and leverage the knowledge graph from the corpus (smart crawling). Hallucination-free, no need for prompt engineering. Zero weight. Suggested alternate prompts based on embeddings.	Poor back-end architecture. Knowledge graph built on top (top-down rather than bottom-up approach). Needs prompt engineering and billions of weights. Yet, the success depends more on auxiliary subsystems rather than on the core DNN engine.
Few tokens: "real estate San Francisco" is 2 tokens. Contextual chunks, KG and contextual tokens with non-adjacent words, sorted n-grams, customizable PMI metric for keyword associations, variable-length embeddings, in-memory nested hashes for KG back-end DB.	Tons of tiny tokens. Fixed-size chunks and embeddings are common. Vector databases, dot product and cosine similarity instead of PMI. Reliance on faulty Python libraries for NLP. One type of token: no KG or contextual tokens.
Focus on conciseness, accuracy, depth and exhaustivity in prompt results. Normalized relevancy scores displayed to the user, to warn him of potential poor answers when corpus has gaps. Augmentation and use of synomyms to map prompt keywords to tokens in backend tables, to boost exhaustivity and minimize gaps. Prompt results (front-end) distillation.	Focus on lengthy English prose aimed at novices, in prompt results. Evaluation metrics do not measure exhaustivity or depth. No relevancy scores shown to the user, or used in model evaluation. No mechanism to reduce gaps other than augmentation. Back-end distillation needed to fix poor corpus or oversized token lists.

Specialized sub-LLMs with LLM router. User can choose categories, agents (built in the back-end), and negative keywords. Or fine tune front-end intuitive parameter in real-time, with debugging option. Process prompts in bulk. Fine tune back-end parameters. Popular user-chosen parameters used for self-tuning, to generate default parameter sets. No training needed. Parameters local to sub-LLM, or global.	User interface limited to basic search box, doing one prompt at a time. No real-time fine-tuning, little if any customization available: the system guesses user intents (the agents). Fine-tuning for developers only, may require re-training the full model (costly), and it is based on black-box parameters rather than explainable AI. Needs regular training as new keywords show up and the model is not trained on them.
Use of multi-index and deep retrieval techniques (e.g. for PDFs). Highly secure (local, authorized users). Can connect to other LLMs or call home-made apps (NoGAN, LLM for clustering or predictions). Taxonomy and KG augmentation. Pre-made template answers with keyword plugging to cover many prompts.	Single index. Proprietary and standard libraries may miss some tables, graphs and other elements in PDFs: shallow retrieval. No KG Augmentation. Data leakage; security and liability issues (hallucinations). Long answers favored over conciseness and structured output.

Index