(automatic-prefix-caching)=

# Automatic Prefix Caching

## Introduction

Automatic Prefix Caching (APC in short) caches the KV cache of existing queries, so that a new query can directly reuse the KV cache if it shares the same prefix with one of the existing queries, allowing the new query to skip the computation of the shared part.

:::{note}
Technical details on how vLLM implements APC can be found [here](#design-automatic-prefix-caching).
:::

## Enabling APC in vLLM

Set `enable_prefix_caching=True` in vLLM engine to enable APC. Here is an example:

```python
import time
from vllm import LLM, SamplingParams

# A prompt containing a large markdown table. The table is randomly generated by GPT-4.
LONG_PROMPT = "You are a helpful assistant in recognizes the content of tables in markdown format. Here is a table as follows.\n# Table\n" + """
```

ID	Name	Age	Occupation	Country	Email	Phone Number	Address
1	John Doe	29	Engineer	USA	john.doe@example.com	555-1234	123 Elm St, Springfield, IL
2	Jane Smith	34	Doctor	Canada	jane.smith@example.com	555-5678	456 Oak St, Toronto, ON
3	Alice Johnson	27	Teacher	UK	alice.j@example.com	555-8765	789 Pine St, London, UK
4	Bob Brown	45	Artist	Australia	bob.b@example.com	555-4321	321 Maple St, Sydney, NSW
5	Carol White	31	Scientist	New Zealand	carol.w@example.com	555-6789	654 Birch St, Wellington, NZ
6	Dave Green	28	Lawyer	Ireland	dave.g@example.com	555-3456	987 Cedar St, Dublin, IE
7	Emma Black	40	Musician	USA	emma.b@example.com	555-1111	246 Ash St, New York, NY
8	Frank Blue	37	Chef	Canada	frank.b@example.com	555-2222	135 Spruce St, Vancouver, BC
9	Grace Yellow	50	Engineer	UK	grace.y@example.com	555-3333	864 Fir St, Manchester, UK
10	Henry Violet	32	Artist	Australia	henry.v@example.com	555-4444	753 Willow St, Melbourne, VIC
11	Irene Orange	26	Scientist	New Zealand	irene.o@example.com	555-5555	912 Poplar St, Auckland, NZ
12	Jack Indigo	38	Teacher	Ireland	jack.i@example.com	555-6666	159 Elm St, Cork, IE

13	Karen Red	41	Lawyer	USA	karen.r@example.com	555-7777	357 Cedar St, Boston, MA
14	Leo Brown	30	Chef	Canada	leo.b@example.com	555-8888	246 Oak St, Calgary, AB
15	Mia Green	33	Musician	UK	mia.g@example.com	555-9999	975 Pine St, Edinburgh, UK
16	Noah Yellow	29	Doctor	Australia	noah.y@example.com	555-0000	864 Birch St, Brisbane, QLD
17	Olivia Blue	35	Engineer	New Zealand	olivia.b@example.com	555-1212	753 Maple St, Hamilton, NZ
18	Peter Black	42	Artist	Ireland	peter.b@example.com	555-3434	912 Fir St, Limerick, IE
19	Quinn White	28	Scientist	USA	quinn.w@example.com	555-5656	159 Willow St, Seattle, WA
20	Rachel Red	31	Teacher	Canada	rachel.r@example.com	555-7878	357 Poplar St, Ottawa, ON
21	Steve Green	44	Lawyer	UK	steve.g@example.com	555-9090	753 Elm St, Birmingham, UK
22	Tina Blue	36	Musician	Australia	tina.b@example.com	555-1213	864 Cedar St, Perth, WA
23	Umar Black	39	Chef	New Zealand	umar.b@example.com	555-3435	975 Spruce St, Christchurch, NZ
24	Victor Yellow	43	Engineer	Ireland	victor.y@example.com	555-5657	246 Willow St, Galway, IE
25	Wendy Orange	27	Artist	USA	wendy.o@example.com	555-7879	135 Elm St, Denver, CO
26	Xavier Green	34	Scientist	Canada	xavier.g@example.com	555-9091	357

```
Oak St, Montreal, QC     |
| 27  | Yara Red      | 41  | Teacher       | UK            | yara.r@example.com     | 555-1214      | 975 Pine St, Leeds, UK       |
| 28  | Zack Blue     | 30  | Lawyer        | Australia     | zack.b@example.com     | 555-3436      | 135 Birch St, Adelaide, SA   |
| 29  | Amy White     | 33  | Musician      | New Zealand   | amy.w@example.com      | 555-5658      | 159 Maple St, Wellington, NZ |
| 30  | Ben Black     | 38  | Chef          | Ireland       | ben.b@example.com      | 555-7870      | 246 Fir St, Waterford, IE    |
"""


def get_generation_time(llm, sampling_params, prompts):
    # time the generation
    start_time = time.time()
    output = llm.generate(prompts, sampling_params=sampling_params)
    end_time = time.time()
    # print the output and generation time
    print(f"Output: {output[0].outputs[0].text}")
    print(f"Generation time: {end_time - start_time} seconds.")


# set enable_prefix_caching=True to enable APC
llm = LLM(
    model='lmsys/longchat-13b-16k',
    enable_prefix_caching=True
)
```

```
sampling_params = SamplingParams(temperature=0, max_tokens=100)

# Querying the age of John Doe
get_generation_time(
    llm,
    sampling_params,
    LONG_PROMPT + "Question: what is the age of John Doe? Your answer: The age of John Doe is ",
)

# Querying the age of Zack Blue
# This query will be faster since vllm avoids computing the KV cache of LONG_PROMPT again.
get_generation_time(
    llm,
    sampling_params,
    LONG_PROMPT + "Question: what is the age of Zack Blue? Your answer: The age of Zack Blue is ",
)
```

## Example workloads

We describe two example workloads, where APC can provide huge performance benefit:

- Long document query, where the user repeatedly queries the same long document (e.g. software manual or annual report) with different queries. In this case, instead of processing the long document again and again, APC allows vLLM to process this long document *only once*, and all future requests can avoid recomputing this long document by reusing its KV cache. This allows vLLM to serve future requests with much higher throughput and much lower latency.

- Multi-round conversation, where the user may chat with the application multiple times in the same chatting session. In this case, instead of processing the whole chatting history again and again, APC allows vLLM to reuse the processing results of the chat history across all future rounds of conversation, allowing vLLM to serve future requests with much higher throughput and much lower latency.

## Limits

APC in general does not reduce the performance of vLLM. With that being said, APC only reduces the time of processing the queries (the prefilling phase) and does not reduce the time of generating new tokens (the decoding phase). So APC does not bring performance gain when vLLM spends most of the time generating answers to the queries (e.g. when the length of the answer is long), or new queries do not share the same prefix with any of existing queries (so that the computation cannot be reused).