1. Describe how your prefetcher works.

I use a 16-entry stream table to spot repeating memory strides. Each entry tracks:

- Last accessed address
- Detected stride (positive or negative)
- Confidence counter
- Valid bit

On every memory load:

1. I check if it matches a stream's expected address.
2. If not, I grab a free entry or replace one in round-robin order.
3. I compute stride = current_address – last_address.
4. If the stride matches the previous one, I would add more confidence; otherwise I reset it.

When confidence reaches 2, I start prefetching, scaling up to 4 lines ahead based on how steady the pattern is. This keeps useful data flowing for regular accesses while avoiding wasted fetches on random or one-off touches.

2. Explain how you chose that prefetch strategy.

I chose a stride-based prefetcher with confidence tracking because it is good at handling regular access patterns, like fetching data in loops or arrays. This is common in many applications. The dynamic prefetch distance keeps things efficient—aggressive when patterns are clear (repeated), but conservative when they're new (the first time fetching). Its simple stream table tracks multiple streams in parallel.

3. Discuss the pros and cons of your prefetch strategy.

Pros:

- Effective and simple for predictable stride patterns
- Adaptive distance with confidence mechanism to limit waste
- Tracks multiple streams in parallel

Cons:

- Ineffective for irregular accesses (e.g., pointer chasing)
- Fixed, small table may evict useful streams under pressure
- No feedback on prefetch usefulness, risking cache pollution

4. Demonstrate that the prefetcher could be implemented in hardware (this can be as simple as pointing to an existing hardware prefetcher using the strategy or a paper describing a hypothetical hardware prefetcher which implements your strategy).

This design is used in real CPUs:

- IBM POWER5's stride prefetcher
- Intel DCU's IP-based stride unit
- AMD K8's stride prefetcher

Hardware needs are minimal:

- 16-entry SRAM table (~12 B/entry)
- Simple ALU for stride math
- Comparators to match streams
- Counters for confidence

These are standard, low-area blocks in modern processors.

5. Cite any additional sources that you used to develop your prefetcher.

- Chen T. & Baer J. (1995). Hardware-based data prefetching. IEEE Trans. Computers, 44(5):609–623.
- Pugsley S. et al. (2014). Sandbox prefetching. In HPCA '14.