API used in SoMe Application with arguments and reasonings:

The choice between Request-Driven (RD) and Request-Response (RR) APIs, for instance, for a social media application, depends on many factors. These factors relate to the type of data interaction involved, performance needs, flexibility desired, and so on. Following is the breakdown of each API paradigm in the context of a social media application and how each could apply, basically:

1. Request-Driven API REST

REST, or Request-Driven API: This is based on resource-oriented architecture. Every resource may be accessed by a unique URL, using standard HTTP methods like GET, POST, PUT, and DELETE.

Social Media Applications:

• There is a simple and clear structure: REST is easy to learn and implement, and thus the best solution

for basic social media CRUD operations like posting, updating, or deleting any content.

• Caching: REST APIs can use HTTP caching mechanisms to improve performance, which helps in social media feeds or profiles that don't really need real-time updates.

Stateless Architecture: Since in REST every request is independent and contains all the data required, rest allows much more scalability and loose administration over the distributed systems.

Cons for Social Media Applications:

Over-fetching/Under-fetching Issues: REST can suffer from over-fetching/under-fetching problems because it normally returns a full resource, even when parts might be needed. For example, a performance issue in social media apps would occur during the loading of complex data, like profiles with several posts, comments, and reactions.

Less Flexibility regarding Data Requirements: Most social media applications need data ranging across multiple resources. For example, profile data is often needed with posts, comments, and reactions. REST is not that efficient in these complex requirements of fetching data; it may require several requests to fetch all the data.

2. Request-Response API: GraphQL

In a Request-Response API, like GraphQL, the clients can request only what they need. In one request, the client describes what the data structure looks like and gets just that data from the server, regardless of how complex the underlying data is.

Social Media Applications:

Fetching Data Efficiently:

GraphQL saves developers from over-fetching and under-fetching by allowing the clients to detail the exact data they want. This is particularly useful in social media applications where users browse content with many levels of data, like for example, a user profile that has multiple posts, comments, and reactions.

• Flexibility and Rapid Iteration: GraphQL lets front-end developers change the requirements regarding data without making any changes on the back end, which ultimately increases the pace of feature development and optimization of user experience.

• Complex Data Relationships: Social media applications very often need data from multiple related resources-for example, to show the user's post along with comments, reactions, and associated media. GraphQL can easily handle these complex relationships within one query.

Cons for Social Media Applications:

- Site Complexity of Setup and Maintenance: With GraphQL, the setup is more complicated and hence it might raise development and also maintenance time.

- Caching Challenges: GraphQL, unlike REST, does not implement any kind of HTTP caching mechanism out of the box, and this may affect its performance in conditions when data does not change frequently.

Choosing Between the Two for Social Media Application

Indeed, GraphQL would be more effective for most modern social media applications, considering its flexibility and efficiency when working with nested data structures and complex sets of relationships. Social media applications are bound to serve efficiently, especially when user interactions grow increasingly complex. However, there are situations that would call for REST:

• Simple Use Cases: When the application is simple or has few interdependent resources, a REST API would suffice and may be easier to implement. • Caching Needs: In cases where there is a frequent need for caching of data, such as static images or general information, the use of REST's HTTP caching may be beneficial. Conclusion

NOTE: However, both API paradigms have their strong points, and for an application like a social media app that requires dynamic data and nested data, GraphQL would come in even more handy when optimizing data fetching, complex relationships handling, and front-end flexibility. It could make use of REST for those aspects of an application that require strong caching or simple and isolated operations.