The MVP is a proof of concept for a search engine for images. It processes images and responds to user queries. However, the MVP ranks and serves queries on a single process, the results are stored in memory and there is no way to rank additional images. Any failure would result in a complete outage and loss of data. Rather than placing every resource on a single server, the resources should be divided into individual components.

The ranking process would benefit from separation. At the moment, most of the time is spent waiting for API responses. This could be improved upon by sending concurrent requests. But for a large set of inputs, throughput becomes a problem. Separating the ranking process allows the throughput problem to be handled with horizontal scalability. In addition, a server outage from the ranking process would not bring the search engine service down. Separating the ranking process makes the service capable of handling greater throughput and failed nodes.

Separating the data storage prevents the loss of data through redundancy and persistent memory. The current implementation keeps every result on one machine's volatile memory. A better way to keep memory is in persistent storage. This does introduce latency when the system wants to read, write or update image rankings. To decrease this latency, a persistent cache could introduced. If the rankings require low latency, a persistent cache could potentially replace the database. This would be possible assuming that each image returns an average of ten concepts, each url averages to 100 characters and each list has about 50 bytes of overhead. The total outcome would come out to 1.58TB in the worst case. Having three replicas would bring the total to 4.74TB. Separating the storage gives great flexibility for latency while providing durability.

The query process would benefit in a manner similar to the ranking process. Separating the query process gives the advantage of horizontal scalability. With horizontal scalability comes fault-tolerance and greater throughput. Users will see fewer outages and reduced loading time as a result.

Separating the ranking, data and query components yields a faster, durable system. This analysis focused on the benefits achieved through a better system design. This is because the largest issues are the design itself. More implementation inefficiencies exist. However, these are due to my inexperience with Go so they were left out of the analysis. Also, the current implementation only uses the general model for image analysis. This implementation works as a proof of concept and those features could be considered for the future.