Eloquent is Laravel's implementation of Active Record pattern and it comes with all its strengths and weaknesses.

Active Record is a good solution for processing a single entity in CRUD manner - that is, create a new entity with filled properties and then save it to a database, load a record from a database, or delete.

You will benefit a lot from Eloquent's features such as dirty checking (to send SQL UPDATE only for the fields which have been changed), model events (e.g. to send administrative alerts or update statistics counters when someone has created a new account), traits (timestamps, soft deletes, your custom traits) eager/lazy loading etc. You can also apply domain-driven pattern and implement some pieces of business logic in your Active Record entities, for example, validation, managing relations, calculations etc.

But, as you already know, Active Record comes with some performance price.

When you process a single record or a few records, there is nothing to worry about. But for cases when you read lots of records (e.g. for datagrids, for reports, for batch processing etc.) the plain Laravel DB methods is a better approach.

For our Laravel based applications we are using both approaches as we see appropriate. We use Laravel's Eloquent for UI forms to process a single record and use DB methods (backed by SQL views with additional database engine specific performance tweaks) to retrieve data for UI tables, export tasks etc. It also works well with RESTful APIs - Eloquent for GET, PUT, POST, DELETE with a key and DB for GET without key but with filters and sorting and paging.

Laravel Eloquent vs query builder — Why use eloquent to decrease performance.

Eloquent is Laravel's implementation and it comes with all its strengths and weaknesses.

Eloquent is a good solution for processing a single entity in CRUD manner — that is, create a new entity with filled properties and then save it to a database, load a record from a database, or delete.

You will benefit a lot from Eloquent's features such as dirty checking (to send SQL UPDATE only for the fields which have been changed), model events (e.g. to send administrative alerts or update statistics counters when someone has created a new account), traits (timestamps, soft deletes, your custom traits) eager/lazy loading etc. You can also apply domain-driven pattern and implement some pieces of business logic in your Active Record entities, for example, validation, managing relations, calculations etc.

But, as you already know, Eloquent comes with some performance price. When you process a single record or a few records, there is nothing to worry about. But for cases when you read lots of records (e.g. for datagrids, for reports, for batch processing etc.) the plain Laravel DB methods is a better approach. For our Laravel based applications we are using both approaches as we see appropriate.

We should use Laravel's Eloquent for UI forms to process a single record and use DB methods to retrieve data for UI tables, export tasks etc. It also works well with RESTful APIs — Eloquent for GET, PUT, POST, DELETE with a key and DB for GET without key but with filters and sorting and paging.

Why Laravel Eloquent:

Executing a query in an OOP manner. No binding with table schema. i.e. Even if you change your table name, no need to touch a single query(there may have 1000 queries) to make it work. Just change the table name in the eloquent model. The relationship among tables can be maintained in an elegant way. Just mention the type of relationship, nothing else(JOIN, LEFT JOIN, RIGHT JOIN etc.) needed in query anymore to get data of related tables. Queries are highly readable while written using Eloquent comparing with Query Builder.

Database tables are often related to one another. For example, a blog post may have many comments, or an order could be related to the user who placed it. Eloquent makes managing and working with these relationships easy. (From Laravel Official docs.)