

Known Limitations



- [Performance Considerations](#)
 - [Static Site Generation \(SSG\)](#)
 - [Recommended Solutions](#)
 - [Caching Limitations](#)
 - [Host-Specific Caching Support](#)
 - [Alternative Caching Approaches](#)
 - [Technical Boundaries](#)
 - [Pagination Limits](#)
 - [Connection Model Limitations](#)
 - [Feature Boundaries](#)
 - [Core WordPress Features Not Supported](#)
 - [Features Requiring Extensions](#)
 - [Following Development](#)
 - [Feature Requests and Bug Reports](#)
 - [Contributing](#)
-

This guide outlines known limitations of WPGraphQL and provides guidance on working within these constraints.

Performance Considerations

Static Site Generation (SSG)

A common issue users encounter is when trying to statically generate all possible pages at build time. This can lead to self-inflicted Denial of Service (DOS) when fetching data for hundreds or thousands of pages simultaneously.

For example, trying to generate all posts, pages, products, and their variations at build time:

```
// ⚠ This approach can overwhelm your WordPress server
export async function generateStaticParams() {
  const allPosts = await fetchAllPosts()    // 100s of posts
  const allPages = await fetchAllPages()     // 100s of pages
  const allProducts = await fetchAllProducts() // 1000s of products
  // ... more fetches
}
```

Recommended Solutions

1. Incremental Static Regeneration (ISR)

- Generate the most important pages at build time
- Let other pages build on-demand with ISR
- Set appropriate revalidation periods

```
// ⚡ Better approach using ISR
export default async function Page({ params }) {
  // This page will be generated on first request
  // and revalidated every 60 seconds
  const post = await fetchPost(params.id)
}

export const revalidate = 60
```

2. Hybrid Approach

- Statically generate high-traffic pages
- Use ISR for less frequently accessed content
- Consider server-side rendering for highly dynamic content

For more detailed performance optimization strategies, see our [Performance Guide](#).

Caching Limitations

Host-Specific Caching Support

WPGraphQL Smart Cache requires specific hosting infrastructure to work effectively:

- The host must support tag-based caching
- The host must support cache invalidation based on those tags

- The host must properly handle the `X-GraphQL-Keys` response header

Many WordPress hosts don't support this level of cache control, which limits the effectiveness of WPGraphQL Smart Cache. Some hosts that do support the required features include:

- WP Engine
- Pantheon (I believe)
- Servers Running Litespeed Cache (see: <https://gist.github.com/jasonbahl/d777c7229bad5142211a58ed00da6598>)
- Some enterprise hosting providers

Alternative Caching Approaches

If your host doesn't support tag-based caching, you can still implement caching at different levels:

1. **Application-Level Caching**

- Use client-side caching with Apollo Client, Relay, or other GraphQL clients
- Implement page-level caching in your frontend application
- Use static site generation or ISR where appropriate

2. **WordPress-Level Caching**

- Object caching (Redis, Memcached)
- Database query caching
- Transient caching

For more detailed information about caching strategies, see our [Performance Guide](#).

Technical Boundaries

Pagination Limits

WPGraphQL implements cursor-based pagination with some default limits:

- Maximum 100 nodes per request in connections
- This limit helps prevent server overload and ensures reasonable response times
- Can be adjusted using the `graphql_connection_max_query_amount` filter, but increasing it may impact performance

```
# Example of pagination limit
{
  posts(first: 150) { # Will be limited to 100
```

```
nodes {  
  id  
  title  
}  
}
```

Connection Model Limitations

The Relay connection model used by WPGraphQL has some inherent limitations:

- Cannot sort across multiple pages of results (each page maintains its own sort)
- Cannot get total counts without performing a separate database query
- Backward pagination (`last` argument) can be less performant than forward pagination

For more information about working with these limitations, see our [Pagination Guide](#).

Feature Boundaries

Core WordPress Features Not Supported

WPGraphQL intentionally does not support certain WordPress features:

1. Cross-site Multisite Queries

- Each site in a multisite network has its own GraphQL endpoint
- Cannot query multiple sites' content from a single endpoint
- No built-in support for multisite network admin functionality

2. Direct Database Access

- No raw SQL query support
- Must use WordPress APIs and functions
- Custom database tables require proper registration with `$wpdb`

3. WordPress Widgets

- Legacy widget system not supported
- Consider using blocks instead (might require extensions or custom code)

Features Requiring Extensions

Some functionality requires additional plugins:

1. **Advanced Custom Fields Integration**

- Requires [WPGraphQL for ACF](#)
- Supports both free and pro ACF versions
- Allows field configuration through ACF interface

2. **WooCommerce Support**

- Requires [WPGraphQL WooCommerce](#)
- Provides access to products, orders, and other WooCommerce data

3. **Block Editor / Full Site Editing**

- Advanced block data requires [WPGraphQL Content Blocks](#)
- FSE features need custom code or extensions
- Basic block content available as rendered HTML by default

4. **Other Common Extensions**

- SEO plugins (Yoast, RankMath)
- Form plugins (Gravity Forms)
- Custom field plugins (ACF, Meta Box)

For a complete list of available extensions, see the [Extensions Directory](#).

Following Development

Feature Requests and Bug Reports

The best way to stay informed about WPGraphQL development:

1. **GitHub Issue Tracker**

- [Browse existing issues](#)
- [Submit feature requests](#)
- [Report bugs](#)

2. **Release Notes**

- Check the [GitHub releases page](#)
- Major releases include migration guides when needed
- Breaking changes are documented in release notes

Contributing

If you'd like to help address limitations:

1. **Code Contributions**

- Review our [Contributing Guide](#)

- Check the "good first issue" tag on GitHub
- Submit Pull Requests for review

2. **Extension Development**

- Create extensions for missing functionality
 - Share extensions with the community
 - Follow our [Extension Development Guide](#)
-

👤 #1

👤 2026-08-29 15:36:26 UTC

👤 2026-08-29 15:36:26 UTC