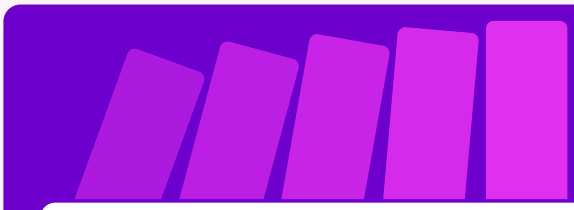
Unit 1 - Tool optimization



Unit 1
Tool optimization

1.1 Unit Introduction

Welcome to the first unit of Tools and AI agent's settings optimization course!

This unit teaches you to optimize tools for your AI agents and apply best practices to make the most out of your tools.

You will learn:

- what is tool optimization
- why optimized tools improve your AI agent's performance
- best practices to optimize your tools

Time to begin!

Continue to 1.2: Tool optimization

1.2 Tool optimization

Tools give AI agents the ability to perform actions, but having tools isn't enough. How those tools are configured determines whether AI agents can actually use them successfully.

When AI agents use tools, they need to:

🎯 **Pick the right tool for the job**
Wrong tool selection leads to wasted calls and incorrect results.

✍️ **Provide the correct inputs**
Incorrect or missing parameters cause the tool call to fail.

📤 **Understand the tool's outputs**
If AI agents can't parse the response structure, they can't use the information.

Handle failed tool calls

When a call returns an error, agents need clear guidance on whether to try again, change parameters, or use a different approach.

How you set up those tools determines whether AI agents can actually use them successfully. This is where tool optimization comes in.

📌 Definition

Tool optimization means designing your tools so AI agents can use them effectively.

Optimizing your tools allows your AI agents to successfully call tools and perform their tasks without getting stuck, making errors, or wasting context window space.

In the Context Engineering course, you learned how to name tools clearly and write descriptions that help AI agents select the right tool.

This course builds on that foundation by covering additional optimization practices when building your AI agent's tools.

Continue to 1.2.1: Why is tool optimization relevant to AI agents?

1.2.1 Why is tool optimization relevant to AI agents?

Poorly optimized tools could cause **failures** that **limit** AI agent performance.

On the other hand, **well-designed tools** enable AI agents to **work autonomously**, **efficiently**, and **at scale**.

✅ Well designed tools

🤖 Higher autonomy

AI agents are able to complete routine tasks independently without human intervention.

💰 Lower costs

Reduce token usage by returning only necessary data and preventing failed tool calls.

📈 Scalability

Maintain consistent performance and provide reliable responses, eliminating the need for

❌ Poorly optimized tools

📊 Too much data

The tool returns more information than needed, wasting the AI agent's context space on irrelevant data.

❓ Confusing inputs

Unclear requirements cause the AI agent to provide wrong or incomplete values, leading to failed tool calls.

⚠ Unclear errors

Error messages don't explain what failed or how to fix

Continue to 1.2.2: Best practices for optimizing your AI agent's tools

1.2.2 Best practices for optimizing your AI agent's tools

This unit covers three key practices for optimizing your AI agent tools:

1. Managing tool verbosity
2. Input and output design
3. Handling errors in your tools

These practices work together to make your tools easier for AI agents to use. When you apply them, AI agents can perform their tasks reliably, use fewer tokens, and work more independently. You will look at each of them in detail.

Let's start with the first practice: managing tool verbosity.

Continue to 1.3: Managing tool verbosity

1.3 Managing tool verbosity

📌 Definition

Tool verbosity refers to the amount of information a tool returns in its response. A verbose tool returns a large amount of information or detail.

Verbosity becomes a concern when tools return more information than the AI agent needs for its specific task.

When a tool returns large amounts of information, it consumes many tokens. More information returned means more tokens used, which fills

the AI agent's context window and increases costs.

How can tool verbosity impact AI agent performance?

See the answer

Unfiltered tool responses can create two performance problems.

Click each one to learn more.

Context window exhaustion

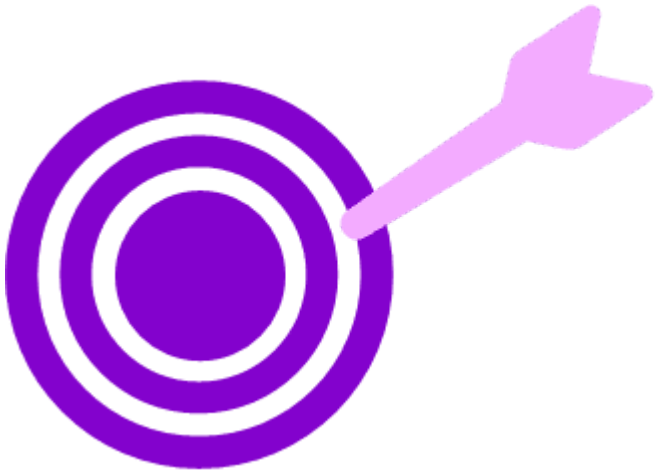
Every token a tool returns takes up space the AI agent needs for other work.

Verbose responses leave less room for reasoning, instructions, call history, and other tool results. This becomes a problem in multi-step tasks where the AI agent must track information across multiple calls.

Decreased accuracy

AI agents process all data you return. When responses include unnecessary information, AI agents take longer to find what they need and may miss critical details.

This could lead to wrong decisions, incomplete tasks, or needing additional tool calls to clarify information that should have been clear from the start.



You can address both problems by configuring **data filtering** in your tools.

When you set up filters, you limit what information reaches the AI agent.

Let's see how to do it!

Continue to 1.3.1: Data filtering

1.3.1 Data filtering

📌 Definition

Data filtering is the practice of limiting tool responses to only the information the AI agent needs, removing unnecessary data before it reaches the AI agent's context.

When you configure data filtering in your tool, you define rules that limit what information gets returned to the AI agent.

Here's how data filtering works:

The AI agent calls your tool.

Your tool retrieves data from a source (database, CRM, API).

It applies the filtering rules you configured, and returns only what's needed for the task.

You control what enters the AI agent's context by configuring your tool to filter and limit data before your tool sends it back to the AI agent.

> When your tool sends back only necessary data, you save context space and help the AI agent focus on completing its task.
>
> To control verbosity, you need to **implement data filtering strategies** so you can reduce unnecessary information.

Let's explore the filtering strategies you can use.

Continue to 1.3.2 Data filtering strategies

1.3.2 Data filtering strategies

When AI agents receive too many results, they must process every record to find what they need.

This wastes context space on irrelevant data and forces AI agents to do filtering work that should happen inside the tool. The AI agent ends up using tokens to sort through information rather than completing its actual task.

When tools return complete records with all available fields, AI agents receive data they don't need.

This wastes context space on irrelevant information. The AI agent must process fields that don't help complete the task.

Tool verbosity can be controlled through **two distinct filtering approaches**. Both reduce the amount of data returned to the AI agent, but they address different aspects of the response.

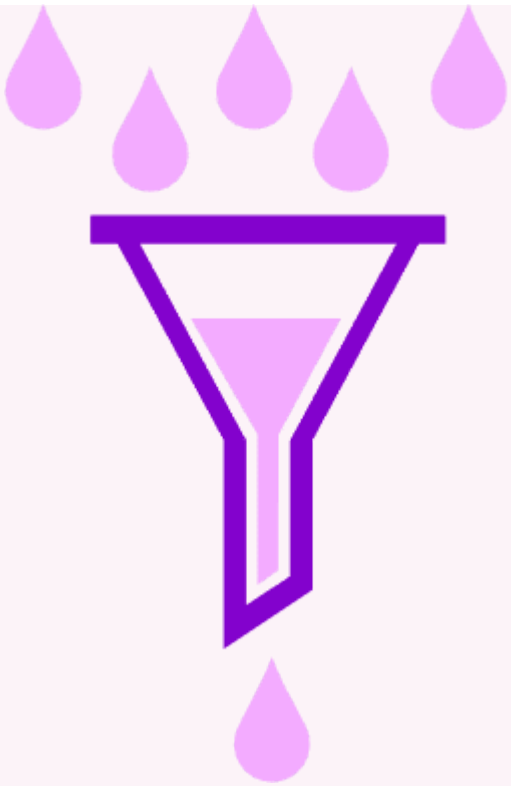
- **Limit the number of results**: filter how many items a tool returns
- **Limit the fields per result**: filter which data fields are included in each returned item

Let's learn more about each.

1: Limit the number of results

1

Limit the number of results



The first type of filtering strategy controls **how many results a tool returns**.

Without filtering, tools can return hundreds or thousands of records, overwhelming the AI agent's context window.

Common filtering parameters include:

- **Result limits**: Set how many items your tool returns. For example, return only 20 results.
- **Date ranges**: Filter by time period. For example, return only orders from the last 30 days.
- **Status filters**: Filter by current state. For example, return only active customers.

- **Geographic filters**: Filter by location. For example, return only customers in Madrid.

These parameters work together to narrow results before they reach the AI agent.

Let's see an example of how to do this **with Make**.

The AI agent's goal is to identify 30 customers from Madrid and send them event invitations.

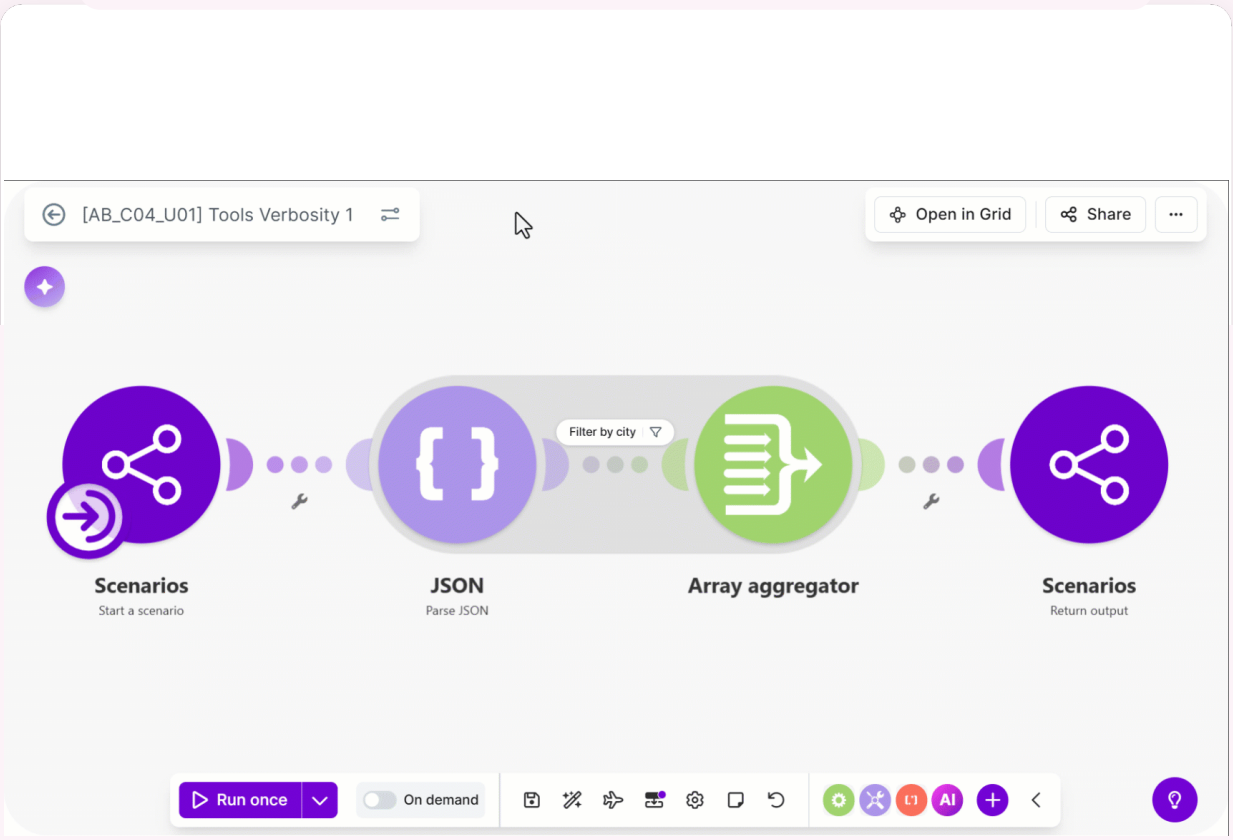
How do you set up filtering parameters in Make?

🟣 **Create a scenario as a tool for your AI agent.**

⚙️ **In your scenario's input settings, define the parameters for city and limit.**

📞 **AI agent calls the tool with values:**

- **City**: Madrid
- **Limit**: 30 results



The tool queries the customer database using these criteria.

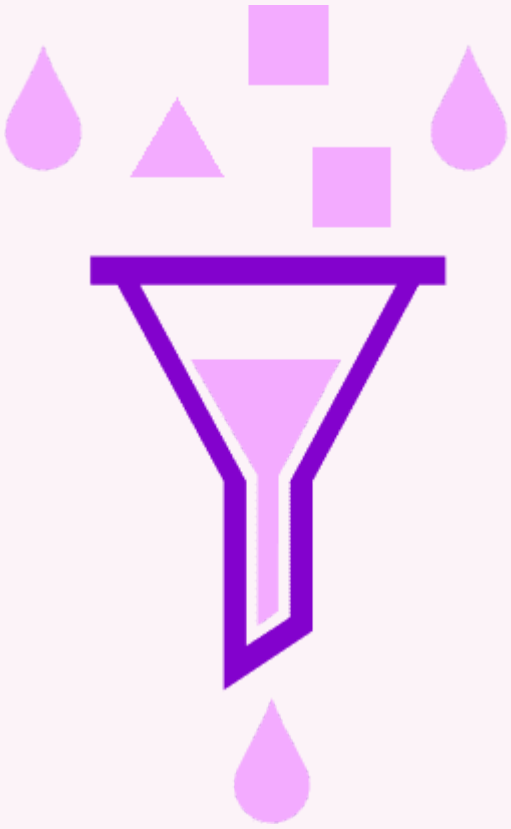
AI agent receives only the 30 relevant customer records, exactly what it needs to complete the task.

---

2. Limit the fields per result

2

Limit the fields per result



The second type of filtering controls **which data fields are included in each returned item**.

Records often contain many fields, but AI agents need only some of them.

> You can configure your tool to return only the fields required for the AI agent's task.

Identify which fields the AI agent actually uses and exclude everything else. This reduces each item's size in the response. Even with the same number of results, fewer fields per result means less data transferred and less context consumed.

Let's discover how you can do this **using Make**.

The AI agent's task is to send event invitations.

It requires only customer contact information, not the complete database record for each customer.

How can you limit fields per result in Make?

🟣 Create a scenario as a tool for your AI agent.

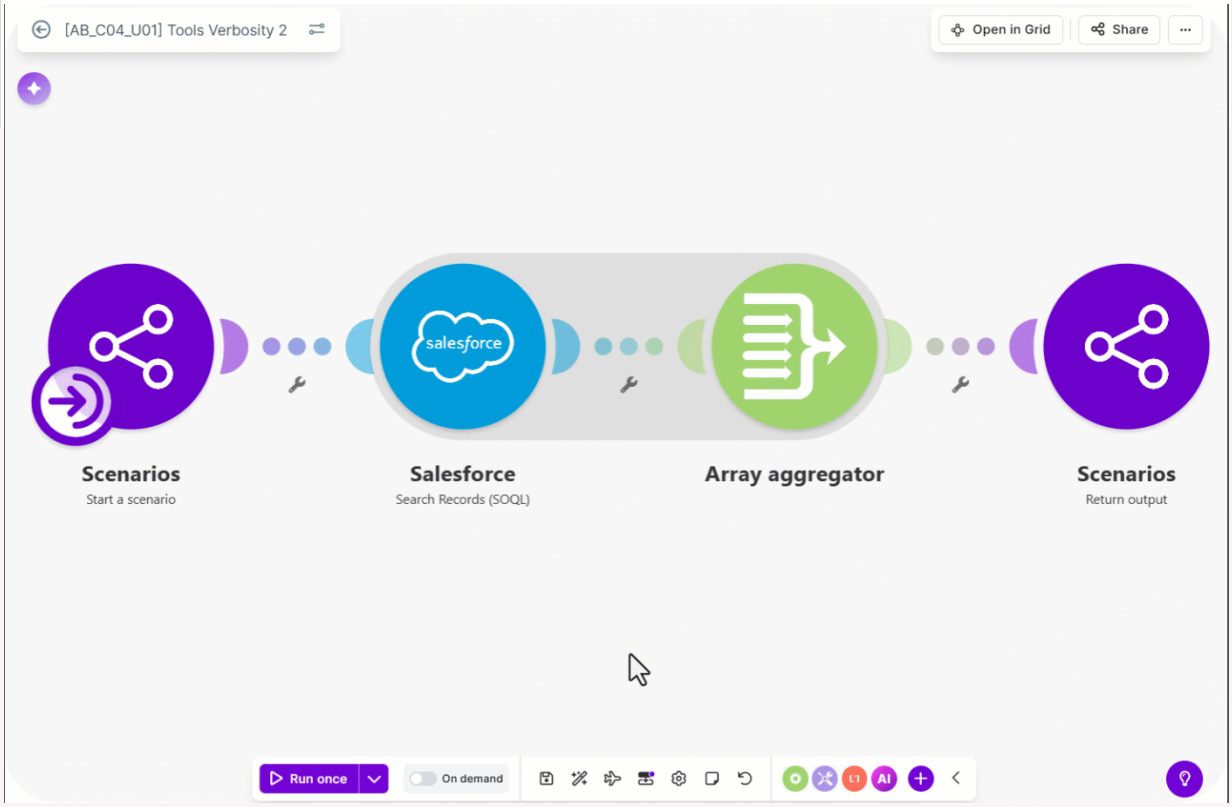
📊 Database record contains:

40 fields including name, email, phone, address, city, birth date, and more.

In your **scenario's output settings**, define the parameters for **first name**, **last name**, **email**, and **city** (4 fields).

Tool excludes:

The other 36 fields that aren't needed for sending invitations.



Despite the database containing multiple fields, the AI agent receives exactly what it needs without unnecessary data cluttering its context.

Pro tip

Filtering only relevant fields not only reduces the size of data returned to the AI agent, it also protects sensitive information.

By excluding unnecessary fields, you prevent exposing critical or confidential data to the LLM that the AI agent is using. This adds a layer of data security to your agentic automation.

If you want to review this in more detail, you can go back to the Information management and security course.

By filtering both the number of results and the fields returned, you can control tool verbosity.

Next, you need to clearly structure the data that AI agents send and receive by using inputs and outputs.

Let's explore this in the next section!

Continue to 1.4: Defining inputs and outputs

1.4 Defining inputs and outputs

Configuring your tool's inputs and outputs enables AI agents to use your tools effectively.

Together, inputs and outputs form the data exchange between AI agents and tools:

 **Inputs** make it easy for AI agents to provide correct information.

 **Outputs** make it easy for AI agents to understand and use what they receive.

You covered inputs and outputs in the AI agents in Make course.

Review it if you need a refresher.

Continue to 1.4.1: Input Design

1.4.1 Input Design

Your tool's input parameters determine what information AI agents must provide when calling a tool.

Why do inputs impact tool performance?

See the answer

When inputs aren't properly configured, they cause failed tool calls and wasted attempts.

Let's examine some issues that can come with incorrectly set up inputs.

Common input problems:

❓ **Unclear which parameters are required**
When AI agents don't know which fields are mandatory vs. optional, they skip required information or waste effort on unnecessary fields.

🔢 **Too many input parameters increase errors**
More parameters mean more opportunities for mistakes. AI agents must determine the correct value for each one

the correct value for each one.

📝 **Vague input types cause format mismatches**

When parameters accept any text, AI agents can provide data in many different ways. Without specific type requirements, they might use wrong date formats or provide incomplete values.

How to define tool input parameters?

✅ Require only essential input parameters

Make a parameter required only if the tool cannot work without it. This reduces what AI agents must provide and lowers error rates.

Example: A customer search tool requires **city** (the tool only searches within specific cities). **Customer type** and **limit** are optional, the tool defaults to all customer types and 10 results if the AI agent doesn't specify.

✅ Set default values for optional parameters

When a parameter isn't critical, make it optional and provide a default. This reduces what AI agents must provide while keeping the tool functional.

Example: A search tool has an optional **sort order** parameter that defaults to **most recent first**. The AI agent only specifies sort order when it needs something different.

Continue to 1.4.2 Output Design

1.4.2 Output Design

Your tool's output structure determines what data AI agents need to receive and how it's organized.

Why do outputs impact tool performance?

See the answer

When output parameters are not clearly defined, AI agents struggle to understand and use responses effectively.

Common output problems

Inconsistent structure slows processing

When tools return data without a clear description, AI agents must spend extra effort figuring out how to parse the response. This uses up context space and processing time. This could lead to slower task completion and more parsing errors.

Unnecessary fields waste context

Extra fields consume context space. This reduces room available for reasoning and other operations.

How to define tool output parameters?

✅ Separate status from data in outputs

Keep execution information (success/failure, counts, timing) separate from the actual content in your tool's response. Use a status field and a separate data field. This prevents agents from confusing metadata with actual results.

Example: Your customer search tool finds three customers and returns a **status** field showing **success**. Separately, it returns a data field containing the three customer names. The AI agent knows the first parts describe the operation status while the data field contains the actual customers.

As you learned before, you need to configure your tool's response to include only fields the AI agent needs for its task to manage tool verbosity.

Example: Your database stores 40 fields for each customer including **name**, **email**, **phone**, **address**, **birth date**, and **complete purchase history**. When the AI agent needs to invite customers to an event, your tool returns only three outputs: **name**, **email**, and **city**. The other 37 fields aren't relevant for sending invitations.

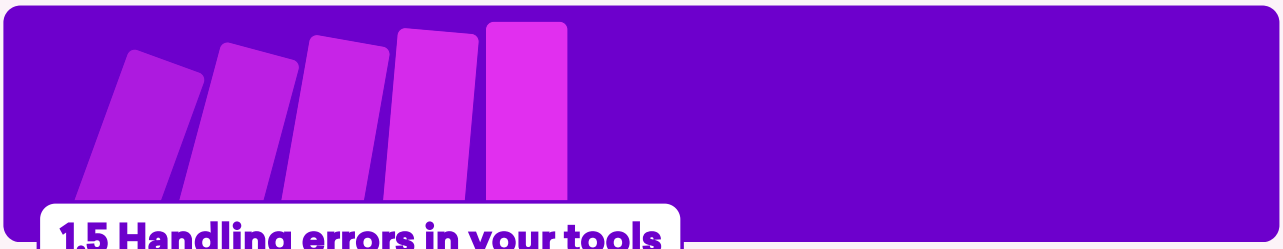
Filtering data and designing clear inputs/outputs control what information flows through your tools.

But even well-designed tools encounter problems, missing data, invalid requests, or system failures.

In the next section, you will examine how to handle errors effectively in your tools.

Let's get to it!

Continue to 1.5 Handling errors in your tools



1.5 Handling errors in your tools

Handling errors in your tools refer to how your tools communicate failures and problems to AI agents.

Errors can happen during tool execution for many reasons, like a missing parameter or an invalid value. When an error occurs, the **tool must return information that helps the AI agent understand the problem and decide what to do next.**

Why is managing errors important for optimizing tools?

🔁 **Prevents retry loops**

Clear errors specify if a retry makes sense, preventing unnecessary attempts.

✅ **Enables task recovery**

When errors explain what's wrong and how to fix it, AI agents correct their approach and succeed on the next attempt.

📉 **Reduces wasted calls**

Specific error messages help AI agents provide correct inputs immediately instead of trying multiple variations.

💾 **Preserves context space**

Failed calls accumulate in the AI agent's context. Good error handling reduces failed calls, keeping context space for actual task completion.

How do you configure error responses for your AI agent tools?

See the answer

When you are setting up your tool, you need to configure what error messages it sends back to the AI agent when something goes wrong.

You set up these messages in advance so when an error happens, your tool automatically returns helpful information instead of generic error codes.

Here are some best practices you can follow:

✅ Be specific about what failed

Don't just share generic error messages, since they force AI agents to guess what's wrong, which leads to multiple failed attempts.

Example: Instead of **Invalid input**, say **Parameter 'email' is invalid. Expected format: user@domain.com but received 'johnsmith'**.

✅ Explain why it failed

Understanding the reason helps AI agents correct the specific issue.

Example: Instead of **Operation failed**, say **Contact was created successfully but birth date was in incorrect format**.

✅ Provide status information for all outcomes

AI agents need to understand what happened even when operations succeed but return no data or partial results.

Example: Return **0 records found matching search criteria** instead of an empty response with no explanation.

✅ Distinguish error types clearly

Different error types require different AI agent actions, so the error needs to be clear.

Example: **Rate limit exceeded, retry in 60 seconds** (retryable) vs. **Invalid email format** (fix parameter, don't retry).

So, what are some do's and don'ts of managing errors in your tools?

✅ Do's

- Be specific about what failed

❌ Don'ts

- Use generic messages

- Explain why it failed
- Provide clear status for all outcomes

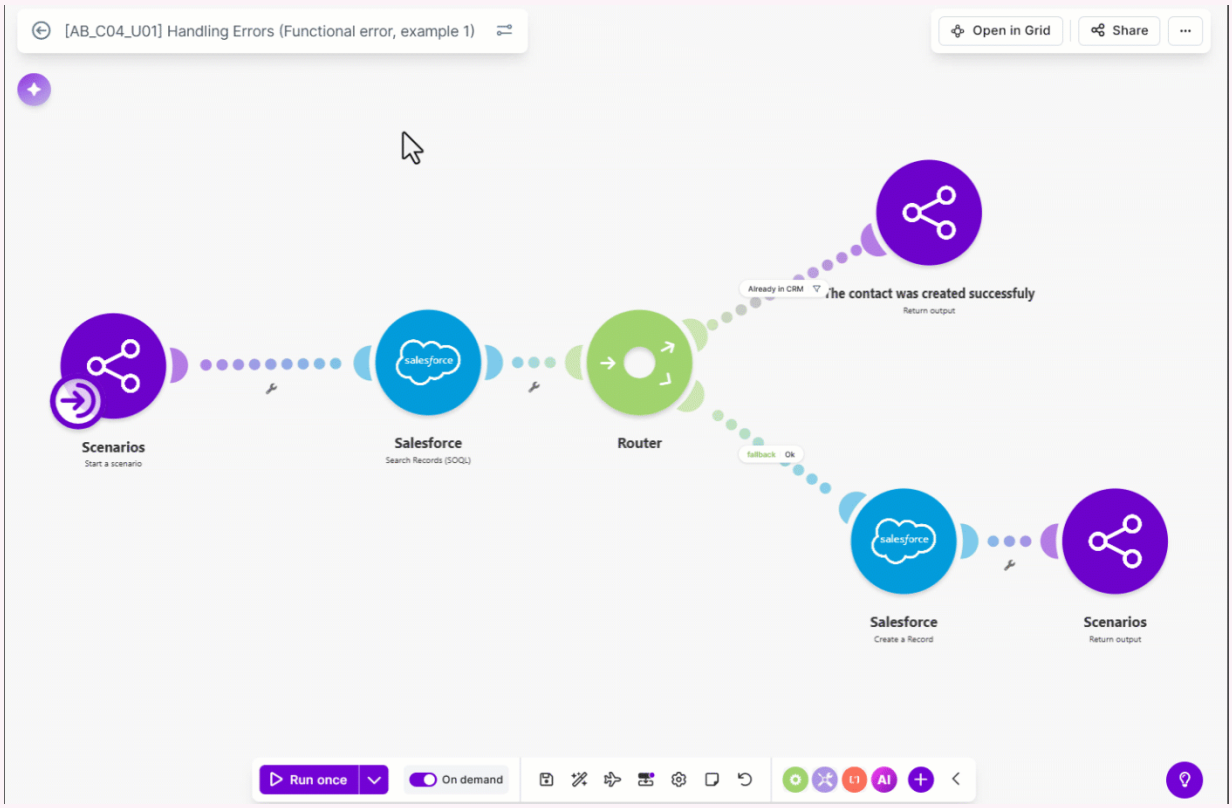
- Give no explanation
- Provide no guidance

When you build a tool in Make, you can set it up to handle potential errors.

Let's take a look at an example of how you can do this.

The AI agent's task is to create and update new contacts in the company's CRM, but it needs to make sure not to duplicate any contact.

How can you identify and prevent an error in your tool using Make?



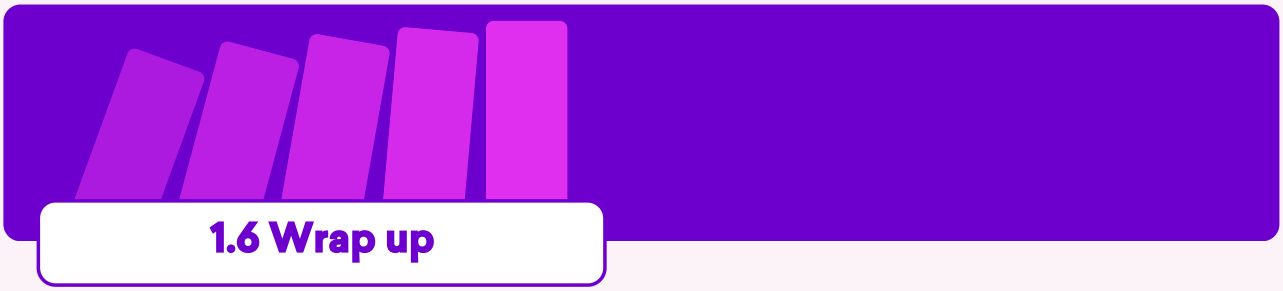
🔎 You add a search module in your tool that checks if the contact already exists in the CRM.

⚙️ You configure what happens when a duplicate is found. You set up the tool to return a specific message: **Duplicate error: Contact with email 'john@example.com' already exists in CRM.**

✅ The **AI agent receives clear information about what went wrong** so it can decide on the next step.

> Combined with filtering strategies and clear inputs and outputs, managing errors in your tools ensure AI agents can navigate failures.
>
> These practices work together to optimize your AI agent's tools.

Continue to the wrap up for this unit



1.6 Wrap up

1

Tool optimization is **setting up your tools so AI agents can use them effectively**. While tools give AI agents the ability to perform actions, how you configure those tools determines whether AI agents can actually use them successfully. Tools that are properly set up deliver **higher autonomy, lower costs, and scalability**.

2

Tool verbosity, the amount of information a tool returns, **directly impacts AI agent performance** by **consuming context space** and **increasing costs**. Data filtering solves this by **limiting tool responses to only what the AI agents need**. Input and output design **control the data exchange**

between AI agents and tools. Well-designed inputs make it easy for agents to provide correct information, while well-designed outputs make responses easy to understand and use.

3

Handling errors in your AI agent's tools supports tool optimization. The information a tool returns to inform of an error needs to be **specific** and **explain why failures occurred**. These practices ensure AI agents can successfully call tools, understand responses, and navigate failures.

Unit complete!

Congratulations!

You've completed the Tool Optimization unit and learned how to set up tools that AI agents can use effectively.

By now you should have an understanding of:

- what is tool optimization
- why optimized tools improve your AI agent's performance
- best practices to optimize your tools

In the next unit, you'll learn how to configure your AI agent's advanced settings.

make | academy

Mark this task complete to continue to the next unit.