Transcript for [Why 99% of AI Automations Fail in Production](https://www.youtube.com/watch?v=Irk4-DO5qgM) by [Merlin AI](https://merlin.foyer.work/)

0:00 - Today I'm going to be talking about five

0:01 - error handling techniques that you need

0:03 - to master if you want to push your

0:04 - workflows into production. It depends on

0:06 - the use case, but sometimes a workflow

0:08 - needs elements of all five of these

0:10 - techniques in order for you to actually

0:11 - be able to set it and forget it and have

0:13 - peace of mind that it's going to run and

0:15 - take care of errors the right way. So

0:17 - I'm going to walk through all five of

0:18 - these. I'm going to show you guys

0:19 - examples. And number four is super super

0:21 - powerful and I feel like it's hardly

0:23 - ever talked about. So you guys will see

0:24 - what I mean by that. So I don't want to

0:25 - waste any time here. Let's hop straight

0:27 - into this workflow so I can show you

0:28 - guys these error handling techniques.

0:30 - Okay, so real quick before we jump into

0:33 - the actual examples I have for you guys

0:35 - here. I wanted to talk about what does

0:37 - production ready mean? So in NN when

0:39 - you're building a workflow and you're

0:40 - kind of in your test environment where

0:42 - you're testing out things and you're

0:44 - changing things and you see the data

0:46 - flow through live, you're in an inactive

0:48 - workflow and as soon as you, you know,

0:50 - flick this switch and you turn this to

0:52 - an active workflow, it basically means

0:54 - okay this is live. If people, you know,

0:56 - whatever your trigger is, your trigger

0:58 - is actively listening now. So if people

0:59 - are emailing you or, you know, sending

1:01 - you a chat in WhatsApp or whatever, the

1:03 - workflow will actually do things and

1:05 - take action in your tools. And that's

1:07 - exactly what an active workflow means.

1:09 - And in order for you to trust that

1:11 - you're ready to go into production and

1:13 - make this an active workflow, there's

1:14 - multiple elements, right? There's like

1:16 - security, there is just the consistency

1:18 - of the outputs and the quality of the

1:19 - outputs in general. But we're focusing

1:21 - today on the air handling aspect, which

1:23 - really is for the peace of mind. Because

1:25 - imagine if you didn't have proper error

1:26 - handling set up and you weren't getting

1:28 - notifications and things weren't, you

1:29 - know, continuing down the path. You

1:31 - could wake up to like 2,000 fails and

1:34 - the whole logic is wrong and you're

1:36 - missing all these different orders and

1:37 - stuff like that. So production ready

1:39 - error handling in my mind means you have

1:41 - a workflow that when it errors, it's

1:43 - sending you notifications. It's logging

1:45 - all of those errors. It has retry and

1:47 - fallback logic and when it fails, it

1:49 - fails safely. Meaning it's not emailing

1:51 - thousands of people or deleting records

1:53 - from your database or inserting a ton of

1:55 - records in your database. And the reason

1:57 - why you have to plan for those failures

1:58 - is because the failures are inevitable

2:00 - in a production environment. Things will

2:02 - fail. But near the end, we're going to

2:03 - talk about building guard rails. And in

2:05 - order to build guard rails, you need to

2:07 - know what type of failures are coming.

2:08 - As much as you can predict edge cases,

2:10 - you don't know what you don't know. And

2:12 - things are going to happen. So by

2:14 - tracking and logging all of your errors,

2:15 - you can start to identify patterns. And

2:18 - when you start to identify patterns, you

2:19 - can build guard rails against those

2:21 - patterns. So anyways, that's enough

2:23 - blabbering from me. Let's just move on

2:25 - right here to the first type of error

2:27 - handling that we're going to talk about,

2:28 - which is kind of like the lowest hanging

2:31 - fruit and the easiest one to set up.

2:32 - Every single workflow you have should be

2:34 - pointing to some sort of error workflow,

2:36 - which you know that's number one, error

2:37 - workflows. So what an error workflow is

2:39 - is it's a separate workflow that starts

2:41 - with an error trigger. And the error

2:43 - trigger can link up to any of your

2:45 - active workflows. And the idea is that

2:47 - whenever an active workflow errors, it

2:49 - will just notify this workflow and then

2:51 - you can set up the logic of what do I

2:53 - want to happen with an error. So I'm not

2:55 - going to go I'm not going to dive into

2:56 - how you actually set this up. I linked a

2:58 - full video on my YouTube channel right

2:59 - here if you guys want to check that out.

3:01 - But let's say Mr. Bad agent here has an

3:04 - Air Table tool and the Air Table

3:05 - credential all of a sudden expires or

3:07 - the scope changes or something. If we

3:09 - weren't having this agent pointing to

3:11 - this error workflow, we would basically

3:13 - have no idea that thousands of records

3:16 - are erroring and we would have no idea

3:18 - what happened later because we would

3:19 - come back and check our workflow and

3:21 - we'd see we have all these errors, but

3:22 - we wouldn't know the error message. So,

3:24 - it'd be really hard to debug. So,

3:26 - because we can set up the logic to do

3:27 - whatever we want, we can go ahead and

3:28 - check the error logger, we can get our

3:30 - notification and fix it as soon as

3:32 - possible. So, that's number one, error

3:34 - workflows. Number two is the ability to

3:36 - have our workflows retry on failure.

3:38 - What this means is that whenever our

3:41 - node faces an error, it's just going to

3:43 - try again. And you can control like wait

3:45 - this much time and then try again, try

3:47 - again five times and then just move on,

3:49 - whatever you want. And the use case here

3:51 - is, you know, sometimes a server might

3:52 - just have some temporary downtime or

3:54 - sometimes there's a little bug.

3:56 - Sometimes it is a good thing to just

3:58 - make sure your workflow nodes will just

4:00 - retry and the way you do that is within

4:03 - any node. So, like an AI agent node,

4:05 - you're going to go up to your settings,

4:07 - and you can see right here, you can turn

4:08 - on the switch that says retry on fail.

4:10 - So, I'll turn that on. And you can see

4:12 - it now opens up these two other things

4:13 - that say max tries, how many times you

4:16 - want it to retry, and then how long do

4:17 - you want it to wait between tries. So,

4:19 - you have a couple levers here to pull in

4:22 - order to change the way that the logic

4:24 - of this retry works. And like I said,

4:26 - it's not just an AI agent node. It's

4:27 - basically any node in NN. So, something

4:29 - like a Gmail API, you can also do a

4:32 - retry and fail. And then even Naden's

4:34 - core nodes that don't even really use

4:35 - like a different server or um AI at all

4:39 - like a code node you can have it retry

4:40 - on fail HTTP request like basically any

4:43 - node in here can retry and fail. So

4:45 - that's a really easy sort of like you

4:47 - know low barrier to entry type of retry

4:50 - failure you could do. There is another

4:52 - kind of like more advanced technique

4:53 - called polling which isn't exactly retry

4:55 - failure but kind of and I'll kind of

4:57 - touch on that in the guardrail section

4:58 - at the end of the video. But let's move

5:00 - on to number three, which is having a

5:02 - fallback LLM. So let's set the scene

5:04 - here. We have a fallback agent with open

5:06 - router as the brain. Let's say we come

5:08 - in here and we basically want it to

5:09 - retry on fail three times. Okay. But

5:13 - what happens is I set up this open

5:14 - router credential with a fake key. So

5:16 - it's not going to work. So when I go

5:18 - ahead here and I try to chat to the

5:19 - agent, what's going to happen is it's

5:21 - going to fail. It's going to try again.

5:23 - It's going to try again. And it tried

5:25 - three times before it gave us the error

5:26 - message of, you know, invalid

5:28 - credential. So what we can do is have

5:30 - even another error handling technique in

5:32 - here where we have our retry but we can

5:34 - also do a fallback model. So if I check

5:36 - this on it basically allows us to

5:38 - connect a different model in case the

5:40 - main one fails. So let's say open router

5:42 - is our favorite and we're using GPT 4.1

5:44 - mini. Open router's down or open AI is

5:47 - down. What we can do is we can connect

5:48 - another model over here and we can just

5:50 - go with Google Gemini. And now we have

5:52 - this model in place. So now if I save

5:55 - this workflow and I say hi, it's going

5:57 - to try that first model, it fails. And

5:59 - then what happens is it just goes to the

6:01 - fallback model. And now we make sure

6:03 - that we're still at least getting some

6:04 - sort of answer. And if you don't see

6:06 - that fallback model option, I think it

6:07 - was a new release of Naden 1.101

6:10 - somewhere around there. So go ahead and

6:12 - update it and then you should see that.

6:14 - All right, so number four, like I said,

6:15 - this one's my favorite one and I feel

6:16 - like it's not talked about very often,

6:18 - but this is the ability to have your

6:20 - nodes continue on an error. So there are

6:22 - times when you may have like some

6:23 - fallback logic or whatever it is, but

6:26 - for some reason something just isn't

6:28 - working, but what you don't want to

6:30 - happen is for your entire flow to stop.

6:32 - So think about this example. Every

6:34 - morning you have like a thousand new

6:36 - entries to process and you want to do

6:37 - some research and you want to I don't

6:39 - know write some sort of content. What

6:41 - happens is if you're going to like loop

6:42 - through all a thousand of those runs and

6:44 - the first item fails, then the rest of

6:48 - the 999 will not get processed. But if

6:51 - you can have it just continue even if it

6:53 - errors, then maybe 998 of them are good,

6:56 - two are bad, but at least you're not

6:58 - just left sitting there with absolutely

7:00 - nothing. So let me show you guys a quick

7:02 - example of that. We've got this code

7:04 - node. Don't worry about the code node.

7:05 - Basically, I just told this to output

7:06 - three different values for us. Google,

7:08 - meta, Nvidia. It's going to output those

7:10 - items and loop through them and do

7:12 - research on them using Tavali. And

7:14 - what's going to happen is I have the

7:15 - third one set up to error the body

7:17 - request. So, if I run this real quick,

7:20 - we're going to see it pull in those

7:21 - three items. It's going to loop through.

7:23 - The first one's going to go fine. The

7:24 - second one's going to go fine. And now

7:25 - the third one is going to error. And

7:27 - what happens is it stopped the flow. So,

7:29 - if we had 20 more to process, it

7:30 - wouldn't process them. And just in case

7:32 - you guys are curious, the reason why it

7:33 - stopped the flow is because I had the

7:36 - actual value being passed over with

7:39 - quotation marks. So, if you guys know

7:40 - like a JSON body, if you have double

7:42 - quotes, it's going to break that

7:43 - request. So you guys can see if I'm at

7:45 - run one, it was fine because the search

7:46 - query was Google with no doubled

7:48 - quotations. Run two was fine, but run

7:50 - three failed and that had the double

7:52 - quotations around Nvidia. So what we can

7:55 - do is change the setting in this HTTP

7:57 - request to Tavi to continue even if that

8:00 - one of the runs fails. So I can click

8:02 - into here, I can go to settings, and all

8:04 - I have to do is change the on error,

8:06 - which is by default to stop the whole

8:07 - workflow. And we just switch that to

8:09 - continue. So it's the exact same flow.

8:11 - I'm going to execute this. It's going to

8:13 - pull in those three items. It's going to

8:14 - loop through them. The first one's good.

8:16 - Second one's good. The third one fails,

8:19 - but it doesn't fail and stop the whole

8:20 - workflow. So, if I go into the Tavly

8:22 - node now, we can see that the first two

8:24 - ran, right? They have their search

8:26 - results, and the third one just

8:28 - basically sent an error message, which

8:29 - was JSON parameter needs to be valid

8:31 - JSON, but it still followed the rest of

8:33 - the loop. And if there were 20 more, the

8:34 - remaining 20 would have got processed as

8:36 - well. And then if we want to get even

8:38 - more robust to maybe track things that

8:41 - didn't work in a separate one, we can

8:43 - actually do one more thing where if I

8:45 - move this trigger down here, we can have

8:48 - the errored items go down a separate

8:50 - branch, which is a continue on error.

8:53 - And so you guys may have seen this in

8:54 - some of my other videos where I have

8:55 - some agents doing this to log, you know,

8:57 - different outputs based on um if I was

9:00 - successful or if the agent fails, it's

9:01 - going to do this. But let me show you

9:03 - guys an example real quick. In the

9:04 - settings of Tavi, we just changed the on

9:06 - error operation to continue using an

9:09 - error output which creates an extra

9:10 - output. And now if I trigger this guy,

9:13 - it's going to run those exact same three

9:15 - queries. The first one's going to be

9:16 - fine, second one's going to be fine,

9:18 - third one's going to error, but not stop

9:19 - the workflow. And now you can see

9:22 - basically what happened is we had two

9:24 - items go down the success branch. Um,

9:25 - where's the two? Right here. I can't

9:28 - hover over it, but it's, you know, right

9:29 - in the middle. And then we have one item

9:31 - go down the error branch. So I can click

9:32 - into this node. We can have the error

9:34 - item was Nvidia, but the two success

9:36 - ones were um Google and Meta as you can

9:41 - see. So now we're able to maybe feed in

9:45 - this path. We can create some new logics

9:47 - like we can send ourselves an email and

9:48 - say okay here are the variables that

9:50 - errored or something like that just so

9:52 - we still have that tracking going on.

9:55 - Okay, so number five is polling that I

9:57 - kind of alluded to earlier. I said it

9:59 - was going to be in the guard portion but

10:00 - I made it its own thing, right? So

10:02 - polling is basically a technique where

10:03 - we're going to check in on the status of

10:04 - something until it's done. So here the

10:07 - example that I have for you guys is

10:08 - we're going to make one request to PI

10:10 - API to generate us an image using AI.

10:13 - What happens is we hit PI API server and

10:15 - we say hey I want an image and I want it

10:16 - to look like this. And then they

10:18 - basically start working on that item.

10:21 - And what happens is we have a different

10:22 - request that we need to make in order to

10:24 - get the item back. But if we make a

10:26 - request and the item isn't done, it

10:28 - would basically just we wouldn't have

10:29 - our image, but we would still continue

10:30 - down the rest of the workflow. So what

10:32 - we do is use a technique called polling.

10:34 - So let me show you guys what that looks

10:35 - like. I'm going to hit execute workflow.

10:38 - And basically it's going to make a post

10:39 - request to PI API to create that image.

10:42 - And then we check in right here if the

10:44 - image is done. And the image is not

10:45 - done. So now you can see we're going to

10:47 - wait until the image is done. So it's

10:49 - waiting. It checked back again. Still

10:50 - not done. It checked back a third time.

10:52 - Still not done. And we're literally just

10:53 - going to sit here and keep pulling until

10:56 - the image actually comes back. And

10:58 - that's how you can make sure you don't

10:59 - have to play around with like, do I wait

11:01 - 30 seconds or what's the average time?

11:02 - Do I wait a minute? This lets us make

11:04 - sure that we only continue down the true

11:06 - branch with the rest of the automation

11:08 - once it's done. So, if you watch any of

11:10 - my like faceless shorts videos or

11:12 - anything like that, we pretty much

11:13 - always use a polling technique to make

11:15 - sure that our assets are ready to go.

11:17 - So, you can see here it had to check a

11:19 - total of eight times and on the eighth

11:21 - time the image was done. So, if I click

11:22 - into this little expression right here,

11:24 - we can see that we actually do have our

11:26 - image URL. Let me just open this up to

11:28 - prove to you guys it's here. And we have

11:30 - our beautiful picture of a waffle

11:32 - person. And so, real quick, I'll just

11:33 - show you guys how this works. So, like I

11:35 - said, here's the first request. We're

11:37 - making our request to PI API. And we

11:39 - said, hey, we want a picture of a waffle

11:41 - personified as a human. The waffle's

11:43 - wearing a suit and tie. From there, we

11:44 - do an initial weight. So, I put 1 second

11:46 - for the sake of the example, but usually

11:48 - you still want to make this like, you

11:49 - know, 40 or something like that.

11:51 - according to the average wait time.

11:53 - After that, we check in on the status of

11:55 - our request. We send over the task ID,

11:57 - stuff like that. And what you can see is

11:59 - on the first one, we got back a code

12:01 - 200. So, no errors happened. We still

12:03 - got a successful response. But if we

12:05 - scroll down, we can basically see that

12:07 - we don't have a URL. And what we're

12:10 - actually looking for is the status field

12:11 - right here. And it says status equals

12:13 - processing. So, that basically tells us

12:16 - that we're not done yet. The second run

12:18 - status is processing. The fifth run

12:20 - status is processing. But on the eighth

12:21 - one, the status is completed. So all I'm

12:24 - thinking to myself is, okay, how do we

12:26 - make sure we don't move on until the

12:28 - status is complete? Well, what we would

12:30 - do is we have a if node right here,

12:33 - which all we're doing is we're checking

12:34 - if the status equals completed. And when

12:37 - it does, it goes down the true branch,

12:39 - but the previous seven runs, it went

12:41 - down the false branch. And the false

12:43 - branch is just a, you know, a 20 second

12:44 - wait, 5second wait, whatever. And then

12:46 - it goes back. So this is an infinite

12:48 - loop until status equals completed and

12:51 - then we can configure the rest of the

12:52 - logic this way down the true branch. So

12:55 - that's basically how polling works. Just

12:57 - keep in mind that depending on the

12:58 - service you're hitting, you may not need

13:00 - to pull. But if you do need to pull,

13:01 - they're always going to be a little bit

13:02 - different as far as how do you actually

13:04 - do it. So status equals processing and

13:06 - then status equals finished or status

13:08 - equals running and then status equals

13:10 - done. So you kind of have to understand

13:12 - both. what does a in progress run look

13:15 - like and what does a completed run look

13:17 - like and then you can adjust your

13:18 - conditional logic. Okay. And finally, I

13:21 - just wanted to show you guys an example

13:22 - of guardrail. But I wanted to talk about

13:25 - real quick the mindset of error handling

13:28 - and building guardrails. So what I

13:29 - always love to say is that you don't

13:31 - know what you don't know. And when you

13:33 - build a really really robust wire map

13:35 - and you think about your data sources

13:36 - and your transformation, you also

13:38 - probably think about what are things

13:40 - that could go wrong. And you always

13:42 - should assume that more than just that

13:44 - can go wrong because the world is

13:46 - unpredictable and LLMs are unpredictable

13:48 - and you don't know what people may say

13:50 - to your system or you don't know what

13:51 - type of you know inputs your system may

13:53 - get. So, because we have a system where

13:56 - we're able to log all of our errors and

13:58 - full visibility hopefully into the

14:00 - executions, that's where we can sort of

14:02 - identify, okay, when we get results that

14:05 - are bad, why was that? Or when we

14:07 - actually get an error message, why was

14:08 - that? And in automation, your best

14:10 - friend is predictability. So if you're

14:12 - able to spot patterns in certain, you

14:15 - know, errors, that basically creates a

14:17 - little bit of predictability as far as

14:18 - when this type of input comes through,

14:21 - what happens, where does it break, and

14:23 - what can I build to protect against it

14:25 - breaking right there, and then you kind

14:27 - of do. So, so a real quick example

14:29 - that's happened to me in some production

14:31 - workflows frequently, and it's really

14:32 - easy to forget about is just a body

14:34 - request of an API being broken. So,

14:37 - we're going to go back to that Tavly

14:38 - example that we used up here when we

14:40 - were doing this continue on error thing.

14:41 - So, as you guys know, if we send over

14:43 - something to a body request, it's going

14:45 - to fail if it has double quotes or new

14:47 - lines or anything like that. So, what I

14:49 - did here is I'm setting pineapples on

14:51 - pizza with double quotes, and that's

14:53 - being fed into our tablet request down

14:55 - here in the body, which is breaking it,

14:57 - of course, because there's double

14:58 - quotes. And what happens is maybe you

15:00 - have an AI node that's feeding in a body

15:02 - request, stuff like that. And in test

15:05 - environment, it's working because you're

15:06 - not dealing with those double quotes.

15:08 - But if you know that that breaks

15:10 - automations or that breaks requests,

15:12 - then you can just build in a guardrail

15:14 - ahead of time to make sure that that

15:15 - never happens. So we can use this really

15:17 - simple script um expression, whatever

15:20 - you want to call it, where it's

15:21 - literally just saying I'm going to

15:22 - replace any double quotes. And now we

15:25 - have none of them. So as you can see, if

15:27 - I remove that, we have double quotes. I

15:29 - paste that in and now we have no double

15:32 - quotes. So, this basically makes sure

15:34 - that no matter what happens, we're able

15:36 - to have Tavi still process the

15:38 - information every time. And actually,

15:40 - what's pretty cool is if we get rid of

15:41 - this HTTP request because Tavi actually

15:43 - has just a verified community node now,

15:46 - if you didn't know. So, we're connected

15:47 - to Tavi and all I have to do is drag in

15:49 - my search query right here. And what

15:52 - happens is, oh, actually this is

15:53 - connected to the wrong thing. So, I'm

15:55 - dragging in this output into our search

15:58 - query. We have pineapples on pizza with

16:00 - double quotes. But you guys can see I

16:02 - can execute it and it's still going to

16:04 - work because what I'm assuming is going

16:05 - on on the back end is Tavi knew about

16:08 - this and they built in that guardrail

16:11 - when they were creating this, you know,

16:13 - uh, community node because whether I'm

16:15 - feeding over pineapples on pizza with

16:16 - double quotes or I feed it over without

16:18 - double quotes, they both still work. So

16:21 - sometimes if a verified community node

16:23 - or you know a native node is available

16:25 - just go for that one because they might

16:26 - have some of this error handling and

16:28 - guardrails already built into place. And

16:31 - by the way if you guys want to access

16:32 - this entire template for free just to

16:33 - play around with some of this error

16:34 - handling stuff then you can get it like

16:36 - I said for free in my free school

16:38 - community. All you have to do is join it

16:39 - in the link in the description. And once

16:41 - you get in here, just search for the

16:43 - title of the video up here. Or if you

16:45 - click on YouTube resources and go to the

16:46 - title of the video, you'll then be able

16:48 - to see a JSON file and that's what

16:50 - you'll download and import into your NN.

16:52 - And you'll have the exact template right

16:54 - here that we were looking at today. If

16:56 - you like this type of content and you're

16:57 - looking to have some more discussions

16:58 - around stuff like productionready

17:00 - workflows and error handling and

17:02 - surround yourself with a community of

17:03 - like-minded individuals, then definitely

17:05 - check out my paid community. The link

17:07 - for that is also down in the

17:08 - description. We've got an amazing

17:09 - community of over 2,000 members who are

17:11 - building with Nitto end, shipping off

17:13 - production workflows every single day

17:14 - and sharing their learnings and their

17:16 - challenges. We've also got a classroom

17:17 - section with two full courses, agent

17:19 - zero, which is the foundations of AI for

17:21 - beginners and then 10 hours to 10

17:23 - seconds where you learn how to identify,

17:25 - design, and build time animations. So,

17:27 - I'd love to see you guys in those

17:28 - communities. But that is going to be it

17:29 - for the video. If you enjoyed or if you

17:31 - learned something new, please give it a

17:32 - like. Definitely helps me out a ton. And

17:34 - as always, I appreciate you guys making

17:35 - it to the end of the video. I'll see you

17:37 - on the next one.