# The neural-sketch Package: Documentation

Vincenzo Buono

1.10c from 2025/03/22

## Contents

### Abstract

neural-sketch is a modern LaTeX package offering a curated yet customizable environment for crafting publication-ready diagrams, primarily in AI and ML. It supplies *opinionated* defaults for shapes, colors, bridging lines, and group transformations, while enabling deeper fine-tuning via a cohesive key–value system.[1]

[1]This is a preliminary draft. For a more up-to-date version checkout our docs at https://neuralsketch.app/ or code on https://github.com/espressoshock/neural-sketch

# 1 Introduction

## 1.1 Philosophy & Motivation

The neural-sketch package rests on two intertwined principles: an *opinionated* default configuration for generating crisp, publication-ready diagrams, and a *highly customizable* key–value system that grants creative latitude to those who need it. By embracing these two perspectives, it provides authors and researchers with a system that transitions fluidly from rapid, out-of-the-box usage to more intricate, precise diagram design. Through this design, authors can focus on communicating their content without wrestling with endless lines of TeX or TikZ boilerplate.

While neural-sketch particularly shines in AI or machine-learning contexts—where neural network layers, algorithmic flowcharts, and bridging arcs are commonplace—its core principles also extend to more general schematic work. Beneath its user-facing commands, this package is built with LaTeX3 paradigms and the expl3 programming layer, making it robust, internally coherent, and inviting to authors who wish to adapt its internals for specialized or evolving use cases.

## 1.2 Key Features

By default, neural-sketch sets up consistent color palettes, thick yet neat border styles, and built-in shape anchors to produce a professional appearance in minimal lines of code. Overriding any of these preferences is straightforward, allowing you to alter border widths, node fills, or coordinate shifting with a simple key assignment. Consequently, one can generate consistent, rigorously styled, conference-ready figures easily without repeated styling instructions across figures, projects and publications. Internally, the system fuses a modern LaTeX3 toolchain (expl3, l3keys, and l3build) with well-established and familiar TikZ ecosystem. It harnesses `\spath3` library behind the scenes for bridging arcs and advanced path manipulations. It also draws upon `\expl3` modern data structures, property lists and sequences for robust type parsing, key management and configuration.
By default, neural-sketch provides the following modules:

`block` (MODULE)

Offers primitives to draw and style fundamental shapes (e.g., rectangles, circles, diamonds). The `\nskBlock` macro exposes shape-specific parameters for border style, fill, radius, size, and text placement. All shapes from the TikZ ⟨*shapes.geometric*⟩ library are supported, and they can serve as building blocks for more complex diagram elements.

`loader` (MODULE)

Coordinates the loading of optional modules (such as bridges, containers, coords) via `\nskUseModule`. This allows you to load only the functionality you need or quickly enable all available features with `\nskUseModule{`⟨*⟩`}`.

`styles` (MODULE)

Holds global style definitions controlled through a cohesive key–value interface. Users can quickly restyle blocks, containers, and connectors, ensuring consistent visuals throughout a document.

`colors` (MODULE)

Defines and exposes a curated palette of color macros (`\nskBlue`, `\nskGray`, `\nskOrange`, etc.). Authors can readily extend or override this set to match personal or institutional style guidelines.

`groups` (MODULE)

Implements `\nskGroup` for logically grouping multiple shapes or diagram segments under a shared transformation. Typical transformations include shift, rotate, or scale, all governed by a simple key–value syntax.

`containers` (MODULE)

Implements `\nskContainer` to create bounding regions around sets of blocks or shapes, visually separating a subdiagram from the remainder. These containers support fill, borders, corner rounding, and adjustable padding.

`coords` (MODULE)

Enables the placement of named coordinates, optionally with visible markers. Such coordinates act as anchors in diagrams, facilitating precise alignment or adjacency across distinct elements.

`bridges` (MODULE)

Automatically Manages lines, arrows and bridging arcs whenever lines overlap or intersect, using spath3 to split and reorder segments. This module gracefully handles crossing lines in dense diagrams, producing tidy arcs where one path passes over another.

The default color set provides optimized, publication-ready tints (for instance, `\nskBlue`, `\nskGray`, or `\nskOrange`) but can be readily extended. Authors wishing to unify the color scheme across all blocks or containers need only issue a concise `\nskSetStyle` directive, instantly updating the entire document's appearance.

## 2 Loading the Package

```
\usepackage{neural-sketch}
```

To begin, load the package in the preamble of your document:

### 2.1 `\nskUseModule`: Selective Loading of Features

Internally, neural-sketch *only load the core modules and commands.* Optional modules for bridging lines, coordinate helpers, grouping containers, and additional color expansions can be loaded with:

`\nskUseModule` `\nskUseModule {⟨module_list⟩}`

This command loads one or more optional modules from the neural-sketch library. The ⟨*module_list*⟩ is a comma-separated set of module names. Currently recognized modules include `bridges`, `coords`, `groups`, and `containers`. Passing `*` loads all available modules at once.

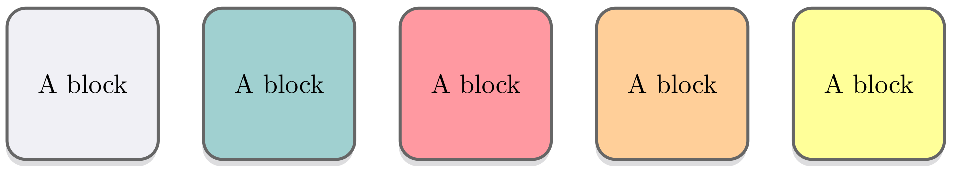
```
% Load two specific modules:
\nskUseModule{bridges, coords}
\nskUseModule{containers}

% Or load everything at once:
\nskUseModule{*}
```

## 3 Getting Started

### 3.1 Minimal Example

A minimal usage scenario involves loading the package and drawing a simple shape with default styles.

```
\begin{nskFigure}
  \foreach \i [count=\x from 0] in
    {nskLightGray, nskBlue, nskRed, nskOrange, nskYellow} {
      \nskBlock[
        type=rectangle,
        id=ablock,
        x={2.6*\x},
        fill=\i,
        text-center={A block},
      ]
    }
\end{nskFigure}
```



## 4 Core Components & Concepts

### 4.1 `\nskFigure`: Creating a Diagram Environment

`\nskFigure` `nskFigure[⟨tikz options⟩]`

The `nskFigure` environment encloses a `tikzpicture` and automatically resets internal counters for `\nskBlock` usage. It also accepts an optional argument, which is passed directly as styling or transformation settings to the underlying `tikzpicture`.

When you begin `nskFigure` with an optional key–value list, these options are applied directly to the underlying `tikzpicture`. Internally, `nskFigure` also clears all `\nskBlock` type counters, ensuring each new figure starts with a fresh naming scheme (such as `rectangle1`, `rectangle2`, and so on).

**TEXhackers note:** Technically, `nskFigure` invokes:

```
\prop_gclear:N \g_nsk_block_counters_prop
\begin{tikzpicture}[#1]
  ...
\end{tikzpicture}
```

Any blocks, coordinates, containers, and connections you define inside this environment inherit the ⟨*tikz-options*⟩ from the optional argument.

The content of `nskFigure` can be freely placed as an in-text figure or wrapped by a standard `figure` environment for floating placement. Either way, `nskFigure` is designed to keep each diagram self-contained, with a fully reset `\nskBlock` counter scope and an optional ⟨*tikz-options*⟩ interface.

### 4.2 `\nskBlock`: Building Blocks

A block is the core unit from which users construct shapes in neural-sketch. Invoking `\nskBlock{`⟨*key-value settings*⟩`}` yields a shape—typically a *rectangle*—that can be relocated, scaled, or annotated. If you do not override any parameters, you receive a rectangle with a subtle fill and a neatly drawn border. You can specify shape type, fill color, border thickness, and text anchors.

`\nskBlock` `\nskBlock [⟨key=value list⟩]`

The command `\nskBlock` serves as the primary mechanism for creating individual shapes (a.k.a. "blocks") in neural-sketch. By default, each block is drawn as a *rectangle* if no further shape information is given. One may, however, specify a shape with the `type` key, for instance:

```
\nskBlock[
  type=diamond,
  fill=nskBlue,
  text-center={Diamond Shape}
]
```

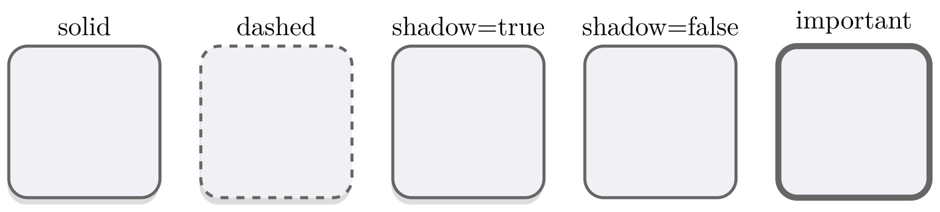
### 4.2.1 Basic styling

`\nskBlock␣(Styling)` `\nskBlock [⟨key = value list⟩]`

In addition to the `type` or geometric shape keys, the user can specify stylistic options for each block. These options include (but are not limited to) changing borders, shadows, or even “importance” (which may scale the border width). Below are three representative keys:

- `border-type`: Accepts `solid` or `dashed` to style the block’s outer boundary.
- `shadow`: A boolean to enable or disable shadows around the block. The default is `true`.
- `importance`: A floating-point multiplier for the border thickness. For instance, `importance=2` doubles the default border width.

```
\begin{nskFigure}[center]
  \nskBlock[
    id=A,
    text-north = {solid},
    border-type=solid,
  ]
  \nskBlock[
    id=B,
    text-north = {dashed},
    border-type=dashed,
    pos={right=.5cm of A},
  ]
  \nskBlock[
    id=C,
    text-north = {shadow=true},
    shadow=true,
    pos={right=.5cm of B},
  ]
  \nskBlock[
    id=D,
    text-north = {shadow=false},
    shadow=false,
    pos={right=.5cm of C},
  ]
  \nskBlock[
    id=E,
    text-north = {important},
    shadow=false,
    importance=2,
    pos={right=.5cm of D},
  ]
\end{nskFigure}
```



### 4.2.2 Supported primitives

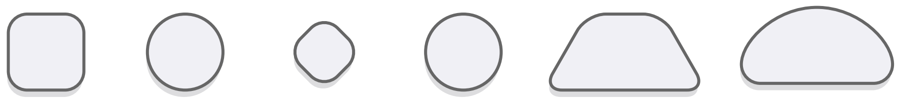
As a default, `\nskBlock` assumes a ⟨*rectangle*⟩ shape if none is specified. However, you can also select from a broader collection of shape names recognized by TikZ's ⟨*shapes.geometric*⟩ library. For instance, users commonly employ `circle`, `diamond`, `ellipse`, `trapezium`, or `regular polygon` shapes. Additional specialty forms such as `semicircle`, `chamfered rectangle`, `cylinder`, and `cloud` are equally accessible.

**TEXhackers note:** The `\nskBlock` macro supports any `TikZ` shape that can be set via `shape=<shape-name>`. A few popular shapes include:

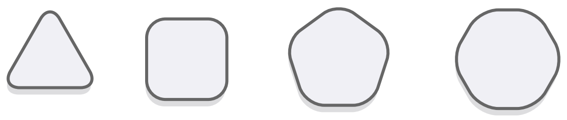
```
rectangle, circle, diamond, ellipse,
trapezium, chamfered rectangle, semicircle,
cylinder, cloud, signal, tape,
regular polygon sides=<n>,
kite, dart, isosceles triangle,
...
```

Some shapes, such as `regular polygon`, allow additional options through `\tikz-opts` (for example, `tikz-opts={regular polygon sides=5}` to produce a pentagon).

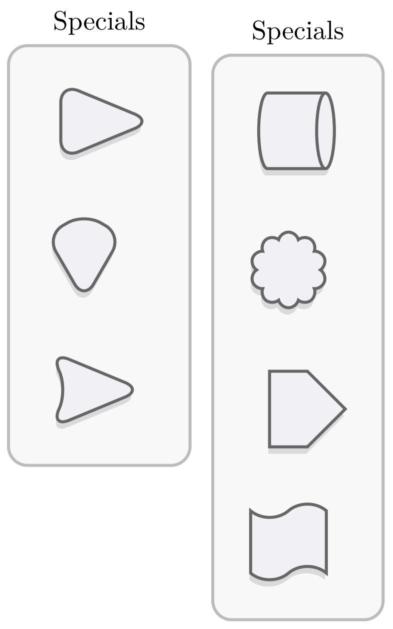
```
\begin{nskFigure}[center]
  \foreach \shape in
    {rectangle, circle, diamond, ellipse, trapezium, semicircle}{
      \nskBlock[
        type=\shape,
        id=ablock,
        width=1cm, height=1cm,
        last-pos-s={right=.8cm}
      ]
    }
\end{nskFigure}
```



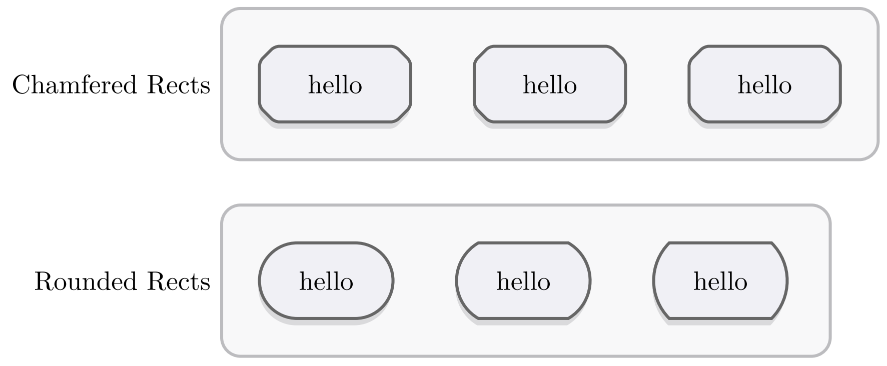
```
\begin{nskFigure}[center]
  \foreach \ns in {3,...,6}{
      \nskBlock[
        type=regular polygon,
        id=ablock,
        width=1.5cm, height=1.5cm,
        tikz-opts={regular polygon sides=\ns},
        pos={right=.8cm of ablock}
      ]
    }
\end{nskFigure}
```



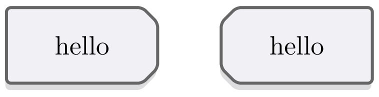
```
\begin{nskFigure}[center]
  \nskCoord[id=ablock, y=0]
  \nskCoord[id=bblock, x=2.7]
  \nskContainer[
    text-north={Specials},
    id=ag,
  ]{
    \foreach \shape in {isosceles
         triangle, kite, dart}{
        \nskBlock[
          type=\shape,
          id=ablock,
          width=1cm, height=1cm,
          pos={below=.8cm of ablock},
        ]
      }
  }
  \nskContainer[
    text-north={Specials},
  ]{
    \foreach \shape in {cylinder,
         cloud, signal, tape}{
        \nskBlock[
          type=\shape,
          id=bblock,
          width=1cm, height=1cm,
          pos={below=.8cm of bblock},
          border-radius=0mm,
        ]
      }
  }
\end{nskFigure}
```



```
\begin{nskFigure}[center]
  \nskCoord[id=ablock, y=0]
  \nskCoord[id=bblock, y=2.6]
  \nskContainer[
    text-west={Rounded Rects},
    id=rrects,
    text-north={},
  ]{
    \foreach \a in {180, 120, 90}{
        \nskBlock[
          type=rounded rectangle,
          text-center={hello},
          id=ablock,
          width=2cm, height=1cm,
          border-radius=0,
          pos={right=.8cm of ablock},
          tikz-opts={rounded rectangle arc length=\a},
        ]
      }
  }
  \nskContainer[
    text-west={Chamfered Rects},
    text-north={},
  ]{
    \foreach \a in {180, 120, 90}{
        \nskBlock[
          type=chamfered rectangle,
          text-center={hello},
          id=bblock,
          width=2cm, height=1cm,
          border-radius=.6mm,
          pos={right=.8cm of bblock},
          tikz-opts={rounded rectangle arc length=\a},
        ]
      }
  }
\end{nskFigure}
```



```
\begin{nskFigure}[center]
  \nskBlock[
    type=chamfered rectangle,
    text-center={hello},
    id=cblock,
    width=2cm, height=1cm,
    border-radius=.6mm,
    last-pos-s={right=.8cm of},
    tikz-opts={chamfered rectangle corners={north east, south east}},
  ]
  \nskBlock[
    type=chamfered rectangle,
    text-center={hello},
    id=cblock,
    width=2cm, height=1cm,
    border-radius=.6mm,
    pos={right=.8cm of cblock},
    tikz-opts={chamfered rectangle corners={north west, south west}},
  ]
\end{nskFigure}
```



You may further customize each shape with `\nskBlock` keys like `width`, `height`, `fill`, `border-type`, and more. Moreover, anything you would ordinarily pass in a `\node[...]{...}` statement can be placed under the `tikz-opts` key of `\nskBlock`, if needed.

## 4.3 ID Generation

When a new block is drawn with `\nskBlock`, its ⟨*id*⟩ key determines how the shape is referenced in subsequent operations, including positioning (`pos=`, `last-pos=`) and bridging. Users may explicitly set:

```
\nskBlock[id=myrect, ...]
```

to assign a custom block ID `myrect`. If none is specified, an ID is *automatically* generated. The automatic name takes the form `<type><counter>`, e.g., `rectangle1`, `diamond2`, etc., derived from the shape's type and a global counter. This counter increments each time a shape of that type is placed, ensuring uniqueness within the same `nskFigure`.

`\nskBlockID` `\nskBlockID`

Expands to the ⟨*id*⟩ of the block currently being drawn. This is particularly useful for styling or annotation that depends on the block's assigned name, for instance to label it or connect to it immediately afterwards.

`\nskBlockIDLast` `\nskBlockIDLast [⟨n⟩]`

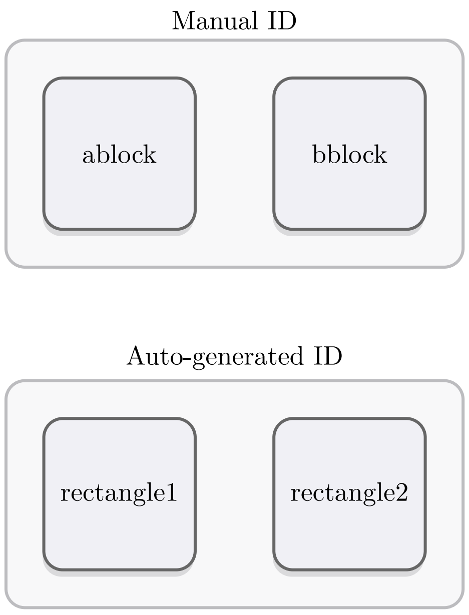
Retrieves the ID of the *n-th last* created block. By default, `\nskBlockIDLast` references the most recently drawn block (`n`=1). Passing, for example, `\nskBlockIDLast[⟨2⟩]` returns the next-to-last block. This command becomes quite convenient in *relative* positioning scenarios, where you wish to specify something like:

```
pos={right=1cm of \nskBlockIDLast[2]}
```

for precise linking with previously created shapes.

**TeXhackers note:** Internally, these IDs (whether user-supplied or auto-generated) are stored in a global sequence, `\g_nsk_block_id_history_seq`, and are reset each time a new `nskFigure` environment begins. Consequently, IDs are scoped *per figure*: once a figure closes, the recorded IDs are no longer carried over.

```
\begin{nskFigure}[center]
  \nskContainer[
    text-north={Auto-generated
         ID},
  ]{
    \nskBlock[
      text-center={\nskBlockID},
    ]
    \nskBlock[
      pos={right=1cm of
           rectangle1},
      text-center={\nskBlockID},
    ]
  }
  \nskContainer[
    text-north={Manual ID},
    shift-y={4.5cm}
  ]{
    \nskBlock[
      % default to
           type=rectangle,
      id=ablock,
      x=0, y=0,
      text-center={\nskBlockID},
    ]
    \nskBlock[
      id=bblock,
      y=0,
      pos={right=1cm of
           ablock},
      text-center={\nskBlockID},
    ]
  }
\end{nskFigure}
```



## 4.4 Positioning Mechanisms

When placing blocks with `\nskBlock`, you have several pathways for specifying their position on the page. These options include straightforward *absolute positioning* with the `x=`, `y=` keys, and more nuanced *relative placement* either via the `pos=` key or by referencing the last-placed block with `last-pos=`.

`\nskBlock␣(Positioning)` `\nskBlock [⟨keys⟩]`

Below are the principal keys for controlling *where* a block appears. They may be combined or overridden according to your layout needs.

**TEXhackers note:** Under the hood, these keys rely on `\expl3` property lists to unify the logic for coordinate parsing, enabling a seamless interplay with the `tikz` ⟨*positioning*⟩ library and the `\nskBlockID`, `\nskBlockIDLast` systems.

`Absolute Positioning` `\nskBlock[⟨x=<fp>, y=<fp>⟩]`

*Absolute positioning* is the simplest approach: specify `x=` and `y=` as floating-point values or zero for the origin. For instance:

```
\nskBlock[
  x=0, y=0,
  text-center={First block}
]
\nskBlock[
  x=3.5, y=2,
  text-center={Moved block}
]
```

Here, the second block appears at coordinates $(3.5, 2)$. This method is straightforward but can become cumbersome in large diagrams requiring repeated relative placements.

`Relative Positioning` `\nskBlock[⟨pos=⟨<anchor spec>⟩⟩]`

When you prefer the high-level ⟨*TikZ*⟩ ⟨*positioning*⟩ syntax, set `pos=...`. Typical usage includes:

```
\nskBlock[
  id=A,
  text-center={Reference}
]
\nskBlock[
  id=B,
  pos={right=1.5cm of A},
  text-center={Relative}
]
```

The block `B` is then placed such that its left edge is 1.5cm to the right of block `A`. You can also specify directions like `below=`, `above=`, `left=`, and so on, or combine them:

```
pos={above right=1cm of XblockID}
```

This makes use of the built-in TikZ positioning library for a concise, readable style.

`Relative (last) Placement`

`\nskBlock[⟨last-pos=⟨<anchor spec>⟩⟩]`

For quick chaining, if your figure *implicitly* wants to place each new block near its predecessor, you may write:

```
\nskBlock[x=0, y=0, text-center={Block 1}]
\nskBlock[
  last-pos={right=1.2cm},
  text-center={Block 2}
]
\nskBlock[
  last-pos={above=0.8cm},
  text-center={Block 3}
]
```

The second block gets placed `right=1.2cm` of *the most recently drawn* block (i.e., `Block 1`). The third block is then placed `above=0.8cm` of *Block 2*. This approach eliminates repeated references to older block IDs and allows for automatic placements.

**TEXhackers note:** Internally, the system checks the final item recorded by `\nskBlockIDLast` to see which block was drawn most recently. If no previous blocks exist, a warning is issued (but compilation continues).

Blocks within a `\nskContainer` or `\nskGroup` can still use the same positioning mechanism. For `\nskContainer`, you may shift or rotate the entire *container and* let each block do local `pos=` or `x=,y=` for its own anchor. Similarly, `\nskGroup` transforms everything in its scope as a unit, but each block's local positioning keys remain valid, yielding consistent results. Minimal example:

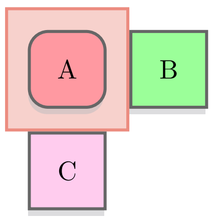
```
\nskContainer[
  fill=nskLightGray,
  shift-x=2,
  shift-y=1
]
{
  \nskBlock[x=0, y=0, text-center={Inside Container}]
  \nskBlock[pos={right=1cm of rectangle1}]
}
```

The container is collectively shifted by $(2, 1)$. Meanwhile, two blocks inside it use standard positioning keys to place themselves *relative* to each other (here via `pos={right=1cm of rectangle1}`).

**Container & Group Implications** When using containers (`\nskContainer`) or groups (`\nskGroup`) with relative positioning keys, neural-sketch implements an internal *phantom reference node* mechanism. In short, these structures do not natively support standard *TikZ positioning* on themselves, so the package's position parser automatically inserts hidden anchor nodes to infer alignment. For instance, if you write `pos={below=1cm of mycontainer}`, the parser interprets the desired anchor (like

`.north` or `.center`) to place the new block. This logic ensures that the block's anchoring edge or corner correctly corresponds to the container's bounding rectangle or group reference. Essentially, neural-sketch sets up ephemeral nodes (`__nsk_phantom_refnode`, etc.) to store the bounding box coordinates. The parser sees something like `below=1cm of mycontainer` and guesses the best anchor (e.g., `mycontainer.north`), then places the block accordingly. In practice, the user can rely on `pos=` or `last-pos=` even for *groups* or *containers*, and the package disambiguates the anchoring behind the scenes.

```
\begin{nskFigure}[center]
  \nskContainer[
    id=ag,
    padding=3mm,
    fill=nskStrongRed!20,
    border-radius=0mm,
    border-color=nskStrongRed!50,
  ]{
    \nskBlock[
      id=A,
      text-center={A},
      fill=nskRed,
      width=1cm, height=1cm,
    ]
  }
  \nskBlock[
    text-center={B},
    fill=nskGreen,
    width=1cm, height=1cm,
    border-radius=0mm,
    pos={right=0cm of ag},
  ]
  \nskBlock[
    text-center={C},
    fill=nskPink,
    width=1cm, height=1cm,
    border-radius=0mm,
    pos={below=0cm of ag},
  ]
\end{nskFigure}
```



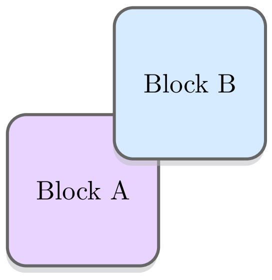
## 4.5 \nskGroup: Logical Grouping & Transforms

When designing multi-block diagrams, you may wish to rotate or shift a collection of shapes as if they were a single entity. The `\nskGroup` command provides a straightforward way to do that. Everything inside its braced content is logically gathered into a group, and transformations (like scaling or rotation) apply uniformly. For instance:

`\nskGroup` `\nskGroup [⟨key=value list⟩] { diagram content }`

Creates a *logical scope* around the enclosed content, applying transformations such as `shift-x`, `rotate`, `scale` to everything inside. A bounding box is tracked (for optional referencing).

```
\begin{nskFigure}[center]
  \nskGroup[rotate=45]
  {
    \nskBlock[
      fill=nskMainAccent,
      text-center={Block A}
    ]
    \nskBlock[
      x=2, y=0,
      fill=nskSecondaryAccent,
      text-center={Block B}
    ]
  }
\end{nskFigure}
```



## 4.6 `\nskContainer`: Wrapped Regions

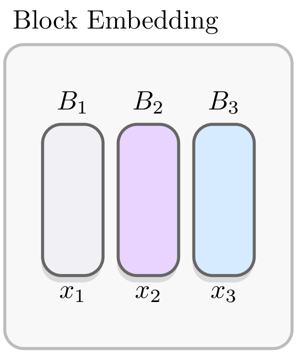
`\nskContainer` `\nskContainer [⟨key=value list⟩] {⟨content⟩}`

This command creates a container or bounding region that encloses the specified ⟨*content*⟩ (for instance, one or more `\nskBlock` commands). Internally, `\nskContainer` leverages `\nskGroup` to group its content under uniform transformations (`shift-x`, `rotate`, `scale`), while also computing and drawing a bounding rectangle that fits the grouped shapes.

**TEXhackers note:** Because `\nskContainer` reuses `\nskGroup` internally, its optional keys can move, rotate, or scale the enclosed items as a cohesive unit, automatically building an invisible bounding box, then rendering a rectangular overlay around that area.

Because the container knows about the bounding box of the grouped elements, it can draw an enclosing rectangle with a specified amount of padding regardless of the enclosed shapes size, type of irregularity. Adjusting `padding=5mm`, for example, leaves extra space between the container border and the shapes inside.

```
\begin{nskFigure}[center]
  \nskContainer[
    text-north={Block Embedding},
    text-north-loc={west},
  ]{
    \foreach \i/\c in {1/nskLightGray,
        2/nskMainAccent,
        3/nskSecondaryAccent}{
       \nskBlock[
         x=\i,
         fill=\c,
         width=8mm,
         text-north={$B_\i$},
         text-south={$x_\i$},
       ]
     }
  }
\end{nskFigure}
```



## 4.7 Patterns

Neural-Sketch supports simple or more elaborate fill *patterns* through two principal keys:

`pattern` `pattern = ⟨TikZ pattern name⟩`

Assigns a standard or custom TikZ pattern to the shape's fill area. Common built-in names include `horizontal lines`, `vertical lines`, `north east lines`, `north west lines`, `dots`, and so forth.

`pattern-color` `pattern-color = ⟨color expression⟩`

Sets the color of the specified pattern. For instance, `pattern-color=nskDarkGray` or `pattern-color=black!50`.

Because TikZ implements these patterns via a `postaction` mechanism, neural-sketch intercepts and appends them to the node style after drawing the shape's fill. If no `pattern` is specified, no patterned fill is generated.

```
\begin{nskFigure}[center]
  \nskContainer[
    text-north={},
    border-type=dashed,
    shadow=false, padding=1.5mm,
    fill=nskLightGray,
    shift-x=-1.2cm
  ]{
    \foreach \i/\p in
      {
        1/north west lines,
        2/north east lines,
        3/vertical lines,
        4/horizontal lines
      }{
        \nskBlock[
          width=1cm, height=1cm,
          id=a, fill=white,
          pattern=\p,
          x=1.2*\i
        ]
      }
  }
\end{nskFigure}
```